HITACHI

MSXベーシック説明書

MSX BASIC Ver. 2.0

8ビットパーソナルコンピュータ

MSX2

はじめに

このたびは日立パーソナルコンピュータをお求めいただき、まことにありがとうございました。
このベーシック説明書では、本機に内蔵されているコンピュータ用のことばであるMSXベーシックver2について説明します。

MB—H3が内蔵しているMSXベーシックは、従来のベーシックがコンピュータの機種ごとに異なっていて、他の機種では同じベーシックでもソフトウェアを共通して使うことができないという不便さを解消しました。
MSX規格のもとで作られたソフトウェアは、どのコンピュータでも共通して使用することができます。
さらにMSXベーシックはver2に改良され、グラフィック機能の拡張、メモリー機能、時計カレンダー機能、漢字処理の追加など、より強力なものになりました。
ゲームやミュージックなどのホビーとしての使用はもちろん、ビジネスや技術計算などの専門的な分野での使用にいたるまで十分な機能と力を持っています。

このベーシック説明書では、MSXベーシックver2の基本的な命令と使い方を基礎編で、またすべての命令や関数の説明を文法編で行います。
MB—H3の機能をフルに発揮させるためにも、本説明書をお読みいただき、正しく末永くお使いください。

● ご 注 意 ●

1．本書の内容の一部または全部を無断で転載することは禁止されています。
2．本書の内容について、予告なしに変更することがあります。
3．本書の内容については、万全を期しておりますが、万一わからない点や誤り、お気付きの点がありましたらご一報くださいますようお願いいたします。
4．運用した結果の影響については、3項にかかわらず責任を負いかねますのでご注意ください。
5．本書に記載した写真、イラストなどは、実際の外観とは一部異なる場合がありますのでご了承ください。

このマニュアルの使い方

全体は3つの部分からできています。

ベーシック基礎編

初めてベーシックを使う方に、MSX2ベーシックを知っていただくために、ベーシックでの命令の書き方やプログラムの組みかたを分かりやすく説明します。
ゲームを中心に本機をご使用になっている方も、ぜひ一度お読みください。プログラムの作りかたを知っていると、ゲームをより楽しく改造したり、自分でゲームを作ることもできるようになります。

ベーシック文法編

すでにベーシックをお使いの方は、この文法編で、MSX2ベーシックの命令や文法をご覧ください。
ベーシック文法編ではこのMSX2ベーシックのすべての命令や関数の機能や書式を機能別に説明します。
プログラムを作るときの辞書替わりに便利なようにアルファベット順の索引も用意してあります。

付録

サンプルプログラムやキャラクターコード、エラーメッセージの一覧表をまとめて載せてあります。
プログラムを作るときの参考にしてください。

MB—H3では、タブレットからの手書き文字入力や画面のハードコピーをベーシックの中でも使用することができます。これらの組み込みの命令の使い方ついては、別冊のMB—H3取扱説明書をご覧ください。
ベーシックでご使用になる場合のこれらの命令は、拡張ベーシック命令としてベーシック文法編でも説明いたします。
SCREEN命令について、VRAMの制約によりMB—H3では使用できない画面モードがありますが、MSXベーシックver 2で標準仕様となっておりますので本文中では説明してあります。これに関して、本文中では特に断りがない場合もありますのでご注意ください。

ベーシック基礎編 目次

Ⅳ ベーシックは便利だ。

Ⅴ 楽しもうグラフィックス&ミュージック

Ⅵ 本格的にプログラミング

ベーシック文法編 目次

一般ステートメント

入出力ステートメント

画面(がめん)・グラフィック ステートメント

音楽演奏(おんがくえんそう)ステートメント

ファンクションキー関係(かんけい)ステートメント

エラー処理(しょり)ステートメント

割り込み処理ステートメント

特殊ステートメント

数値関数

文字関数

特殊関数

特殊変数(とくしゅへんすう)

プリンタ出力命令(しゅつりょくめいれい)

タブレットからの入力命令(にゅうりょくめいれい)

付録 目次

ベーシック基礎編

皆さんがコンピュータに何かをさせようとするときには、コンピュータ用の言葉を使って命令しなければなりません。そのコンピュータ用のことばがベーシックなのです。
ベーシック基礎編では、コンピュータに命令するときの言葉や、命令の仕方を説明します。最初から順に一つ一つ実際に使いながら練習していってください。基礎編が終わるころには、きっとプログラムが作れるようになります。

I　とにかく使ってみよう

コンピュータは難しくありません。
キーの操作になれてしまえば、後はいくつかの命令と便利な関数を覚えれば、楽しく使えるようになります。
積極的にチャレンジしましょう。
操作を間違えても、コンピュータはエラーメッセージで「間違えてますよ」と教えてくれます。
まず、画面に表示させる命令からスタートすることにしましょう。

ベーシックって何？
…まずはじめに

私たちが話をするためには言葉が必要ですね。コンピュータも同じなのです。人間がコンピュータを使う場合、お互いに通じ合う共通の言葉が必要です。このような言葉を「コンピュータ言語」と言います。

人間の使う言葉にも日本語、英語、ドイツ語などたくさんの種類があるように、コンピュータ言語にも用途に応じていろいろなものがあります。本機が持っているコンピュータ言語は「ベーシック」という言葉で、覚えやすく、いろいろな目的に使えるのでほとんどのパーソナル・コンピュータが持っているものです。

ベーシックはアメリカ生まれの言葉ですからちょっと英語に似ていますが、覚えるのはとても簡単です。さあ、ベーシックをこれから学んでいきましょう。

BASIC＝Biginner's Allpurpose Symbolic
Instruction Code

テレビとコンピュータの電源を入れます。
メニュー画面が出てきましたね。
さあ、 F1 キーを押して、ベーシックのスタートです。

画面に何か書いてみよう
…一番やさしいPRINT命令

```
MSX BASIC version 2.0
Copyright 1985 by Microsoft
28815 Bytes free
Ok
■ ←――――カーソル
```

テレビとコンピュータの電源を入れメニュー画面の状態でF1キーを押すと画面は図のようになっていますね。この表示をスターティング・メッセージといい、ベーシックが動きはじめて、いつでも命令をうけつけられる状態になったということを意味しています。字のすぐ下にある白い四角形「■」を**カーソル**といいます。

キーボードからいろいろなキーを押してみましょう。押したキーの通りに画面がでますね。見やすくするために［CAPS LOCK］キーを押して英字の大文字にしましょう。そして、キーを押して

ABC

としてみてください。画面には「ABC」と出るだけで、その横にはカーソルがあるだけで他には何も変わったことはないようです。「ABC」は画面に出ているだけで、コンピュータにはまだ伝わっていないのです。さあ「ABC」をコンピュータに伝えてみましょう。画面に出ている文字をコンピュータに伝えるときには、［RETURN］キーを押します。

ABC ［RETURN］

［RETURN］記号は今後［RETURN］キーを押すことを意味するものとします。何度も出てくるので忘れないでください。

```
Ok
ABC
Syntax error
Ok
■
```

これがコンピュータの返事です。（Syntax error）

「おつぎは何をやりますか」とコンピュータがきいているのです。（Ok）

さて、コンピュータは何か返事をしてくれたでしょうか？あれっ！ピッという音がして画面に何か出てきましたね。

Syntax error

これは「あなたのいっていること（キーボードから打ち込んだこと）はわかりません」とコンピュータが返事をしているのです。「ＡＢＣ」とは人間にしか通じない言葉ですから、コンピュータには理解することができません。だから、コンピュータにわかる言葉で、あなたの言いたいことを伝えなければなりません。コンピュータは決まった形で命令をしてやらないと、全々受けつけてくれないのです。
ピッという音とともに出てくる表示を、エラーメッセージといい、あなたの言いたいことがわからないとか、その通りにすることはできないというコンピュータからの返事なのです。Syntax error の他にまだたくさんのエラーメッセージがありますが、それらはそのたびごとに説明していくことにします。

●PRINT(プリント)

それでは、コンピュータに返事をさせるために一番簡単な命令から覚えていくことにしましょう。それはPRINT命令といいます。

PRINT "オハヨウ" RETURN

と打ち込んでみましょう。最後に RETURN キーを忘れずに押してください。
カタカナを出すときは、一度 かな キーを押してからでしたね。

```
PRINT"オハヨウ"
オハヨウ
Ok
█
```

PRINT "......"?

PRINT "......"は「"」と「"」の間に書いてあるものを画面に書きなさいという意味なんだ

```
PRINT "        "
```

ダブルクォートといいます。

SHIFT ＋ "い 2い で出ますね。

おや？今打ち込んだ文字のすぐ下に“オハヨウ”と出ました。これがコンピュータの返事なのです。PRINTは、「画面に書きなさい」という意味の命令なのですね。このPRINTは基本的な命令なので、この他にもいろいろな使いかたがあります。少しずつ説明していくことにしましょう。

PRINT 2＋3 RETURN

と打ち込んでみましょう。
２＋３の答、つまり５という数字が画面に出てきますね。PRINT命令のあとに計算式を書いておくと、その答を画面に出してくれます。次に、

PRINT "2＋3"

としてみましょう。

今度は２＋３がそのまま画面に出てきました。「"」(ダブルクォート）でかこまれたものをPRINT命令で書くと、それをそのまま画面に出してくれます。PRINT命令を使っていろいろなものを画面に出してみましょう。

```
PRINT "コヤマ"
PRINT 50－30
PRINT "ＡＢＣ"
```

最後に画面にＡＢと書いてみましょう。

```
PRINT AB
 0
```

おや、変ですね。なぜ０になるのかは、後で説明することにします。ここでは、ダブルクォートでかこんで

```
PRINT "ＡＢ"
```

とします。

●PRINTの省略形

PRINTと書くかわりに省略形「？」(疑問符)を使うことができます。

```
? 2 ＋3 RETURN
```

とするとさきほどと同じように答えの5が出ますね。

●間違いの直し方

あなたはキーボードを使うのは初めてですか？もしそうでしたら、キーを押すときに、よく押し間違いをするでしょう。
ここでは、押し間違えたときのなおし方を説明しましょう。

イ）文字の訂正
「PRINT」と打ちたかったのに「ORINT」と打ってしまった。

```
ORINT■
```

カーソルキー[<]でカーソルを「O」のところまでもっていきます。

```
[O]RINT
```

ここで「P」を押します。

```
P[R]INT
```

あとはカーソルを[>]でもとの位置にもどして続きを書きます。

ロ）文字の削除
「PRINT "オハヨウ"」と打ちたかったのに、PRINT "オハハヨウ" と打ってしまった。

```
PRINT "オハハヨウ"■
```

[<]キーでカーソルを「ハ」のところまでもっていきます。

```
PRINT "オハ[ハ]ヨウ"
```

ここで DEL キーを押すとカーソルの「ハ」が消えて、カーソルより右に並んでいた文字が1字ぶん左によります。

```
PRINT "オハ[ヨ]ウ"
```

カーソルをもとに戻してもいいし、1行にこれ以上書かないなら RETURN キーを押します。

ハ）文字の挿入
「PRINT」と打つところを「PRNT」と打ってしまった。

```
PRNT■
```

[<]キーでカーソルを「N」のところまでもっていきます。

```
PR[N]T
```

ここで INS キーを押すとカーソルが下半分の大きさに表示されます。
ここで「I」を押します。

```
PRINT
```

もう一度 INS キーを押すか、[>]キーを押して、カーソルがもとの大きさになるようにします。（ RETURN キーを押したときもカーソルがもとの大きさになります。）

計算だってできるんだ
…たし算、ひき算、かけ算、わり算

PRINT 命令を使って、コンピュータを電卓と同じように使うこともできます。

PRINT　計算式

```
PRINT 5+4
 9
Ok
PRINT 70-52
 18
Ok
PRINT 0.1+5.5-2.3
 3.3
Ok
```

PRINT 計算式は、計算式の結果を画面に書かせなさいという命令なのです。

●たし算、ひき算

たし算、ひき算は普通のとおりに書けば計算してくれます。

PRINT　5＋4 RETURN
PRINT　70－52 RETURN

このとおりに打ち込んでみましょう。ちゃんと計算してくれますね。次に、

PRINT　0.1＋5.5－2.3 RETURN

としてみましょう。たし算、ひき算の混じった計算も、小数点の計算もしっかり答えがでますね。

●かけ算、わり算

かけ算とわり算は、普通とはちょっと違った記号を使います。

かけ算の記号…＊アスタリスクと読みます わり算の記号…／スラントと読みます

それでは計算してみましょう。

PRINT　5＊2 RETURN
PRINT　1／3 RETURN
PRINT　10＊7／2 RETURN

```
Ok
PRINT 5*2
 10
Ok
PRINT 1/3
 .3333333333333333
Ok
PRINT 10*7/2
 35
Ok
■
```

●かっこつきの計算、べき乗の計算

計算式には かっこを使うことができます。

PRINT ((5＋4)＊12＋4) ／2 RETURN

「(」,「)」はいくつ重ねて使ってもかまいません。ただし「(」の数と「)」の数が合っていないと Syntax error になります。

べき乗の計算もできます。2^{10} を計算したければ

PRINT 2^10 RETURN

としてください。

べき乗の記号…^ (指数記号といいます。)

```
PRINT ((5+4)*12+4)/2
 56
Ok
PRINT 2^10
 1024
Ok
■
```

●計算の優先順序

2＋3×5は、25ではなく17だということはわかりますね。かけ算、わり算はたし算、ひき算よりも先に行うのでしたね。こういう規則を「計算の優先順序」といいます。

ベーシックの場合は、次のように決まっています。

$$(\quad) \rightarrow \wedge \rightarrow \left\{ \begin{matrix} * \\ / \end{matrix} \right\} \rightarrow \left\{ \begin{matrix} + \\ - \end{matrix} \right\}$$

次の命令文は何を計算しているかわかりますか？

PRINT 1＋2^4＊5^(1＋2)／2 RETURN

普通の計算式にすると、$1 + 2^4 \times 5^3 \div 2$ となります。答は1001ですよ。

```
Ok
PRINT 1+2^4*5^(1+2)/2
 1001
Ok
■
```

●数の表示

次のように打ち込んでください。

PRINT 100000＊100000＊100000 RETURN

```
Ok
PRINT 100000*100000*100000
 1E+15
Ok
■
```

おやおや？1E+15なんて、おかしな表示が出てきましたね。これは1のあとに0が15個並ぶという意味なのです。

PRINT　1／(100000＊100000) RETURN

としてください。

```
Ok
PRINT 1/(100000*100000)
 1E-10
Ok
█
```

1E－10と出ましたね。これは1の左に0が10個並ぶ(つまり0.0000000001) ということを意味しているのです。

このように、非常に大きな数や非常に小さな数は「E」を使った形で画面に表示されることになります。

●計算の誤差について

コンピュータは、内部で扱う数値を限られたケタ数の中へ収めるために、それ以下の有効数字を四捨五入します。この結果、生じる誤差を丸め誤差とよびます。
たとえば、1÷9×9を計算させますと、

```
PRINT 1/9*9
 .99999999999999
Ok
█
```

というように誤差を生じます。これは、1÷9の答をコンピュータで扱えるケタ数に収めた結果(0.11111111111111)に9を掛けたために生じるのです。ただし、電卓などでは答の最終ケタを四捨五入するものもありますので、上の計算をすると、0.99999999999999の最終ケタが繰り上がり1となります。

●〈起こりやすいエラー〉

●Missing operand(ミッシング オペランド)

かけ算、わり算などで、必要な数を書いていないときに起こります。

PRINT　3＊ RETURN

とした場合などです。「＊」のあとになにか数が必要ですね。

●Division by zero(ディビジョン バイ ゼロ)

0でわり算した場合に起こります。

●Overflow(オーバーフロー)

計算の結果が大きくなりすぎた場合に起こります。エラーが起きない最大の数は

$\underbrace{9.9\cdots9}_{14ケタ}E+62$

負の数も

$-\underbrace{9.9\cdots9}_{14ケタ}E+62$

II プログラムの基礎

プログラムの勉強を始める前に、プログラムを作るのに便利な命令をいくつか紹介します。

プリンタやカセットテープレコーダを使う命令、プログラムの行番号を自動的に発生してくれる便利な命令などです。

プログラムを組む前の準備段階です。しっかり理解しておいてください。必ず役に立ちますよ。

プログラムって何？
…プログラムを作る前に

（CLS）

これまでは、コンピュータに何かさせる場合、まず命令を書いて次に文字や計算式などを書き、最後に RETURN とすればよかったのでしたね。しかし、もっと複雑なことをコンピュータにさせたい場合、これでは大変です。
何をどういう順序で実行すればよいのかが、あらかじめわかっているのなら、そのとおりの順序で命令を覚えさせておけばよいと思いませんか？ 命令を全部覚えさせたあと「実行しなさい」という命令を与えてやれば、1回の指示でコンピュータに連続した仕事をさせることができるわけです。
これが「プログラム」の考え方です。つまりプログラムとは、命令を順序よく並べたものなのです。もちろん、命令はコンピュータが理解できるベーシック言語を使います。

●CLS（クリアスクリーン）

さて、プログラムを作る前に、気分を一新するため、画面をきれいに消してみましょう。そんなとき使うのが **CLS** 命令です。

CLS RETURN

と打ち込んでください。画面に書かれている文字や記号がすべて消えて「 Ok」とカーソルだけが左上にありますね。

画面	
Ok █	CLS 命令を実行したあとの画面です。

SHIFT キーを押しながら CLS HOME キーを押しても同じ結果になります。「 Ok」は出ませんが、画面はきれいになりましたね。

CLS RETURN
（ SHIFT ＋ CLS HOME ）

コンピュータは変数が大好き
…変数、代入文の使い方

PRINT 文で計算できることは説明しました。ここまでは電卓と同じです。コンピュータでは電卓と違う方法で計算することもできます。それは、コンピュータが大好きな変数というものを使うのです。

A＝5 RETURN
PRINT　A RETURN

と打ってみましょう。ちゃんと5という答がでますね。

```
Ok
A=5
Ok
PRINT A
 5
Ok
█
```

どうやら、ここで使っている「A」は数を覚えているようですね。この「A」が変数と呼ばれているものなのです。変数とは数を入れるための箱があり、その箱につけられた名前であると考えてください。

今度は次のように打ち込んでみましょう。

A＝5 RETURN
B＝A＊10 RETURN
PRINT　B RETURN

```
A=5
Ok
B=A*10
Ok
PRINT B
 50
Ok
█
```

答は50ですね。
このように、変数は計算式の中に使うこともできます。ですから、

A＝10 RETURN
A＝A＋1 RETURN

と書くこともできます。A＝A＋1に注意してください。「AはA＋1に等しい」ではありませんよ。「＝」は「右側のものを左側に入れる」の意味です。
ですから、A＝A＋1は「Aに1を加えたものを、もう一度Aの箱に入れる」という意味になります。実際にそうなっているかどうか確かめてみましょう。

PRINT　A RETURN

10に1を加えた11になっていますね。

```
A=10
Ok
A=A+1
Ok
PRINT A
 11
Ok
█
```

変数名 ＝ 式

「＝」(イコール)は「等しい」の意味じゃなくて「右側のものを左側の箱に入れなさい」という意味になります。

変数の名前にはAやBやXなどいろいろなものが使えます。それについては後で説明します。

A＝BとかA＝3とか
こういう形をした命令を
代入文というんです

A＝B

Bの箱の中味を
Aの箱に入れるのか

A＝3

3をいれるんだね

PRINT 変数名

PRINT Aだったら
「Aという名前の箱にはいっている
数字を表示しなさい」
の意味だね

●変数名に使える文字は

さっきの例では、変数名にAやBを使いましたが、もっといろいろな種類の文字を使うことができます。ただし、次の3つの規則を守ってください。

PRINT＝3

変数名の3つの規則

①先頭は英字（アルファベット）で、その後には英字か、数字を使うこと。

例）A，B，ABC，A1，X20，X3Y……変数

2AB……先頭が英字でないので変数でない。

A￥7……「￥」という数字でも英字でもない文字を使っているので変数でない。

②長さは255文字まで。ただし、3文字以上の部分は変数名の区別には使われない。

例）A12345678 7
A12345678 8
A12345678 B

すべて変数だが同じものとみなされる。

最初の2文字だけが区別に使われる。

③ベーシックですでに決められている言葉を使ってはいけない。

例）PRINT 1……PRINT命令とみなされて変数にはならない。

XPRINT……やはり変数には使えない。

PR，XPR……変数として使える。

次のように打ち込んでみましょう。

```
SEIKO=10 RETURN
MASAKI=20 RETURN
PRINT  SEIKO+MASAKI
```

答は30と出ましたか

それでは変数**SEIKO**の内容を確かめてみましょう。

```
PRINT  SEIKO RETURN
```

10となっていますね。

いろいろな変数を作って計算させてみましょう。コンピュータは変数が大好きですから、これからもたくさん変数が出てきます。変数と定数(決まった数字)の計算などをなれるまで練習してください。

ところで、**PRINT**命令で

```
PRINT  AB RETURN
```

としたら、0と表示されましたね。もうおわかりですね。ABが変数として計算されたのです。変数ABには何も入れていませんから、変数ABの中はからっぽ、つまり0と答えてくれたのです。

行番号で順序よく
…行番号の考え方
(NEW, END, RUN)

さあ、やっとプログラムを作るところまでやってきました。その前に、必ずしておかなければならない命令NEWがあります。

●NEW(ニュー)………コンピュータの頭をからっぽにする。
プログラムを作り始める前に必ず

NEW RETURN

としてください。プログラムは、コンピュータに順序よく命令を覚えさせるものでしたね。それで、新しくプログラムを作るときには、前に覚えておいた命令を全部消しておかないと、古いプログラムと新しいプログラムがごっちゃになってしまうのです。
NEW RETURN とすることでコンピュータのなかみはからっぽの状態になります。(電源を入れたときと同じ状態。)
これからは、たくさんの例題プログラムがでてきます。**そのたびにNEW命令を実行して、毎回きれいな状態でコンピュータを使えるようにしておいてください。**

●行番号って何？

さあいよいよプログラム作りです。まずはプログラムに絶対に必要な行番号のお話から。

命令が順序よく並べられたものがプログラムだと言いましたね。さて、その順序とはどうやって決めるのでしょうか？ベーシックでは、命令の先頭に番号をつけることで順番を決めています。たとえば次のように打ち込んでください。

```
10  A=1 RETURN
20  B=5 RETURN
30  C=A+B RETURN
40  PRINT  A RETURN
50  PRINT  B RETURN
60  PRINT  C RETURN
70  END RETURN
```

あれ？なんだかいつもと違いますね。RETURN としても「Ok」が出てきませんし、PRINT 命令の結果も表示されていません。
原因は、先頭に書かれた10~70の数字にあるのです。
1行の最初に数字が書かれると、コンピュータはこの命令を実行せずに、数字の順番を覚えておくのです。
このように先頭に番号のついた命令文の集りを、プログラムといいます。

行番号　命令 RETURN

この、先頭に書く数字を行番号といいます。コンピュータは「実行しなさい」という命令を受けたときに、行番号の小さい順から順序よく実行していきます。

●END(エンド)………プログラムの終わり

行番号70に見かけない言葉がありますね。これをEND命令といいます。「ここまできたらプログラムの実行をやめなさい。」という意味をもつ命令です。

●RUN(ラン)………プログラムのスタート

さあ、プログラムを実行してみましょう。
プログラムを実行させる命令はRUN命令です。次のように打ち込んでください。

RUN RETURN

画面には3つの数字が表示されて「Ok」が出ました。どうやら、ちゃんと計算してくれたようですね。

```
Ok
10 A=1
20 B=5
30 C=A+B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
RUN
  1  ←――― これが行番号40の結果
  5  ←―――      〃     50  〃
  6  ←―――      〃     60  〃
Ok
█
```

●うまく動かなかった人のために

「ピッ」という音がして
　Syntax error in ○○
や
　Missing operand in ○○

などというエラー・メッセージが出た人はもう一度よく画面とプログラムを比較してみましょう。エラー・メッセージの最後の数字がエラーの起こった行番号です。その行のどこかに間違った文字を打っていませんか？その場合には、間違った場所にカーソルを動かして文字を書き直しましょう。書き直しが終わったら、もう一度RUNしてください。今度はうまくいきましたか？もっと詳しい方法は次に説明します。（打ち間違いのなおし方は19ページで説明しましたね。）

〈よく起こるエラー〉

●Syntax error（シンタックス　エラー）

ベーシックにない命令を使うとこのエラーが起きます。たいていは、キーの押し間違いが原因です。

●Missing operand(ミッシング　オペランド)

265ページを見てください。

プログラムを書き換えよう
…プログラムの修正

(LIST)

●LIST(リスト)……プログラムを画面に表示させる。(LISTすると言います。)

前ページのプログラムはうまく動きましたか？こんどは、SHIFT＋CLS HOME キーを押して、画面を消してしまいましょう。あれっ？せっかく作ったプログラムが見えなくなってしまいましたね。でも安心してください。画面から消えてしまってもコンピュータはちゃんと覚えているのです。

LIST RETURN

と打ち込んでみましょう。さっき打ち込んだプログラムを全部画面に書いてくれます。ファンクションキーF4で一度に入力することもできます。

```
Ok
LIST
10  A=1
20  B=5
30  C=A+B
40  PRINT  A
50  PRINT  B
60  PRINT  C
70  END
Ok
█
```

プログラムの一部を見るときには

LIST　始めの行番号－終りの行番号

「－」はマイナス記号を使う。

(「－」は数字キーのとなりの ＝ － のキーですよ。)

LISTの書きかたあれこれ	
LIST	……プログラム全部を画面に表示する
LIST　20－	……行番号20から最後まで画面に表示する
LIST　－50	……最初から行番号50まで画面に表示する
LIST　30－60	……行番号30から60まで画面に表示する

●ちょっと修正してみよう

イ）命令を書き換えたいときは…

行番号30のＣ＝Ａ＋Ｂを

C＝A＊B

に換えたい場合、どうしたらよいでしょうか？方法は簡単です。▷キーや△キーを使って行番号30までカーソルを移動し、「＋」を「＊」に換えればよいのです。換えた後に RETURN とするのを忘れてはいけません。変更したあと RETURN キーを押すとコンピュータは命令を受けつけるのでしたね。

```
Ok
LIST
10 A=1
20 B=5
30 C=A+B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
Ok
■
```

△、▷キーでカーソルをここまで移動し、+を＊に変えた後 RETURN キーを押す。

また、別の方法もあります。改めて

30 C＝A＊B RETURN

と打ってやるのです。

コンピュータは、同じ行番号の命令文が二つあれば、あとから入れた方を覚えてくれるのです。それでは

LIST RETURN

として修正されているかどうか見てみましょう。ちゃんと書き換わっていますね。

```
Ok
30 C=A*B
LIST
10 A=1
20 B=5
30 C=A*B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
Ok
■
```

ロ）命令を1行分消したいとき…

今度は、行番号40と50の命令文を消してみましょう。

40 RETURN
50 RETURN

としてください。行番号だけ押してすぐ RETURN とすると、コンピュータは、その行番号の命令は忘れてしまうのです。**LIST**して確認してください。

```
Ok
40
50
LIST
10 A=1
20 B=5
30 C=A*B
60 PRINT C
70 END
Ok
■
```

行番号40、50はなくなった。

ハ）新しい命令文をつけ加えたいときは…
今度は、行番号60と70の間に

PRINT　A／B

という命令を追加したい場合、どうしたらよいでしょうか。60と70の間に実行されるのですから、

65　PRINT　A／B RETURN

としてやればよいのです。（61～69の間の整数なら、どれでもかまいません。）さあ LIST してみましょう。

```
Ok
65 PRINT A/B
LIST
10 A=1
20 B=5
30 C=A*B
60 PRINT C
65 PRINT A/B  ←ちゃんと60と70の間にはいっています。
70 END
Ok
■
```

このように、プログラムには新しい命令文を付け加えることがあるので、行番号は10番飛びにつけておくのが便利です。もし行番号を１から順番にぎっしりつめて書いておいたら、今のように、新しい命令文を付け加えることはできませんね。

プログラムを保存するには
…セーブ、ロードの方法
(CSAVE, CLOAD, CLOAD?)

コンピュータではプログラムをカセットテープに保存(記録)しておくことができます。また必要なときにもう一度コンピュータの中に読み込むこともできます。このときに使うのが **CSAVE, CLOAD, CLOAD?** 命令です。

プログラムを**CSAVE, CLOAD**する前に、カセットレコーダが本機に正しく接続されているかどうか確かめてください。接続の方法は取扱説明書をよく読んでください。

●CSAVE(シーセーブ)

プログラムをカセットに記録する（これをセーブと言います）命令が**CSAVE**です。

CSAVE　"適当な名前"

「適当な名前」は、記録されたプログラムを、名前で区別するために使います。6文字以内で「"」を含まなければ、どんな文字を使ってもかまいません。なるべく、プログラムの内容が思いだせるような名前を使いましょう。たとえば、記録したいプログラムが、テニスゲームであれば、

```
CSAVE "テニス"
```

とかするわけです。

例として、前ページのプログラムをセーブしてみましょう。プログラムの名前を"TEST"としてみます。

(1) まず、カセットテープを用意して、カセットレコーダに入れます。カセットテープは、一般の音楽用テープでも使えますが、なるべくプログラム用の短かいテープを使いましょう。
音楽用のC−90やC−120などの長時間記録用のテープを使うと、長すぎてテープのどこにプログラムを入れたのかさがすのが大変ですよ。

(2) カセットテープレコーダ TRQ−1500 (別売) は自動的に適正なレベルで記録されますので調節の必要はありません。(取扱説明書をご覧ください。)他のカセットレコーダの場合は、録音レベル調節つまみ（ボリューム）とトーンコントロールを調節してください。

(3) カセットレコーダの録音ボタンと再生ボタンをいっしょに押してください。
TRQ−1500の場合は止まったままですが、リモート端子のないカセットレコーダの場合は動き出します。
(ただし、カセットテープへの記録は始めません。)

(例 TRQ−1500)

(4) **CSAVE"TEST"** と打ち込んでください。

(5) RETURN キーを押します。
リモート端子がついているレコーダの場合、ここで初めてテープが回り始め、プログラムが記録されます。
このとき、テープカウンタの数字をメモしておくとCLOAD(シーロード)やCLOAD？（シーロードベリファイ）するときの頭出しに便利です。

(6) しばらくテープが回ったあと、画面に「Ok」が表示され、カーソルが出てきます。これでプログラムはテープに記録されました。

```
CSAVE"TEST"
Ok
█
```

Okが出るまでの時間は、プログラムの長さによって違います。

リモート端子がついているカセットレコーダはセーブが終ると自動的にテープが止まります。そうでないレコーダは、テープが回り続けます。いずれの場合も停止ボタンを押して停止状態にしてください。

以上で、セーブは終わりです。テープに録音してもプログラムは本機の中に残っています。**LIST**して確認してください。ちゃんと記録されたかどうか心配ですか？それを確かめるために、**CLOAD?** 命令があります。

●CLOAD?（シーロードベリファイ）

CLOAD？命令は、プログラムがカセットテープの中にちゃんと記録されたかどうかを調べる（ベリファイすると言います）命令です。

●CSAVEを行なったあとには、必ずこのCLOAD?を行なって記録がうまくいったことを確認してください。

```
CLOAD?  "プログラム名"
```

「プログラム名」には、ベリファイしたいプログラムの名前を書きます。そのプログラムが見つかるまで、テープは回り続けるので注意してください。
また、「プログラム名」を省略することもできます。その場合、一番最初に見つけたプログラムをベリファイします。

さっきセーブしたプログラムをベリファイしてみましょう。

(1) プログラムを記録してあるカセットテープをカセットレコーダに入れます。記録のときメモしておいたテープカウンタの数を参考にしてプログラムがはいっている場所まで巻きもどしてください。

(2) CLOAD? "TEST" と打ち込んでください。

(3) カセットレコーダの再生ボタンを押し、RETURN を押します。リモート端子がついている場合と、そうでない場合の違いは、CSAVEのときと同じです。プログラムがみつかると「Found：TEST」と表示されカセットテープ（位置出しが終ったらリモート端子の接続をもとどおりにします。）に記録されたプログラムと、本機の中のプログラムとの比較が始まります。

(4) しばらくテープが回ったあと、画面に「Ok」と表示されればプログラムは、正確に記録されていたことになります。「Verify error」と表示された場合は、正確な記録がされていない場合がありますので、カセットレコーダの再生レベルを再度確認してから、もう一度セーブをやりなおしてください。

```
CLOAD?"TEST"
Found:TEST
Ok
```
← 正確に記録されていた場合

```
CLOAD?"TEST"
Found:TEST
Verify error
Ok
```
← 正確に記録されていなかった場合

「Ok」が出れば、ベリファイは終了です。あとは本機の電源を切ったり、NEW命令を実行したりして、プログラムを消してしまっても大丈夫です。ちゃんとカセットテープに記録してあるのですから。

●CLOAD(シーロード)

CLOAD命令は、カセットテープに記録しておいたプログラムを、本機の中に再び読み込む（ロードすると言います）命令です。プログラムを読み込んだ場合、そのとき本機の中にはいっていたプログラムは消えてしまうので注意してください。

```
CLOAD" プログラム名"
```

プログラムを読み込むという点をのぞいて、あとはCLOAD? 命令の操作と同じです。

今度は記録してあるプログラムをロードしてみましょう。まず

NEW RETURN

としてコンピュータの中に残っていたプログラムをすべて消してしまいましょう。NEWしなくともロードはできますが、ここでは、ほんとうに読み込んだことをテストするために、NEWをしてみます。

ベリファイのときと同じ手順で、カセットレコーダを操作してください。キーボードからは、

CLOAD "TEST" RETURN

と打ち込みます。「Ok」が出れば、読み込み完了です。プログラムはコンピュータの中に入りましたか？LISTしてみてください。

```
NEW
CLOAD"TEST"
Found:TEST
Ok
LIST
10 A=1
20 B=5
30 C=A*B
60 PRINT C
65 PRINT A/B
70 END
Ok
█
```

注）以上の命令を実行しても画面に何も返事が戻ってこない場合は CTRL キーを押しながら STOP キーを押して、もう一度最初からよく読んで、やり直してください。

また、取扱説明書「カセットレコーダとの接続」もご覧ください。

プリンタを使おう

(LPRINT,LLIST)

●LPRINT(エルプリント),LLIST(エルリスト)

画面にプログラムを表示させたいときは、LIST 命令でしたね。作ったプログラムを紙に打ち出すと、前後関係がよくわかって便利です。そういうときに、作ったプログラムをプリンタに打ち出す命令がLLIST なのです。画面に何かを表示させたいときには、PRINT命令でしたが、プリンタに何かを表示させたいときには、LPRINT命令です。PRINT命令やLIST 命令と同じ使い方をします。

もしプリンタが接続されていなかったときには、作ったプログラムを一度カセットテープに記録(CSAVEを使うのでしたね)してからコンピュータの電源を切って接続します。接続が終ったら電源を入れ、カセットテープからプログラムをロードしてから、プリンタにプログラムを打ち出します。

●LLIST,LPRINT命令を使うときの注意

LLIST など、プリンタを使う命令を実行するときには、プリンタの接続、用紙がセットされているかどうか、電源が入っているかどうかを確かめてください。プリンタの電源が〝切〟のときに、LLIST 命令などを実行してしまうと、いつまでも命令を実行しようとして待ち続けますので、プログラムはそこで止まったままになります。

III ベーシックを勉強しよう

いよいよプログラムを作り始めます。
簡単なプログラムばかりですから、めんどうがらずにぜひ打ち込んでRUNさせてみてください。
理解できたら数値や文字を変えていろいろ実験してみましょう。
作ったプログラムもできるだけカセットテープにセーブしておくことをお勧めします。
何回も繰り返していくうちに、理解が深まっていくのですから。

はじめに

(AUTO,RENUM,DELETE,REM)

この章から、いよいよ本格的に、ベーシックの命令を説明していきます。
まずは、覚えておくと便利な4つの命令から。

●AUTO(オート)

プログラムを作るには、どうしても行番号を先頭につけなければなりません。プログラムが長くなったりするといちいち打ち込むのは大変手数がかかります。そんなときにこのAUTO命令を使うと便利です。AUTOは行番号を自動的に画面に書いてくれる命令なのです。

AUTO　100，10 RETURN

としてみましょう。はじめの行番号として100を表示してくれました。それではその行番号のところに

100　A＝10 RETURN

と書いて下さい。次の行に110という行番号を書いてくれましたね。どうやら10ずつ増える行番号を自動的に書いてくれるようです。どんどん命令を打ち込んでみましょう。

110　B＝A＊10＋3 RETURN
120　PRINT　B RETURN
130　END RETURN

```
AUTO 100,10
100 A=10
110 B=A*10+3
120 PRINT B
130 END
140
```

ここで CTRL キーを押しながら STOP キーを押す。

このようにして、プログラムを打ち込み終ったら CTRL キーを押しながら STOP キーを押すと、行番号の自動発生が止まります。

AUTO　はじめの行番号,行番号の間隔

はじめに行番号と行番号の間隔を省略してAUTO RETURN とすると、行番号10から10おきに書いてくれます。これから出てくる例題のプログラムにも、このAUTO命令をどんどん使ってください。

AUTO　20……20から10間隔で
AUTO　,5…… 0から5間隔で
AUTO　………10から10間隔で

こんな使い方もあるよ!!

●RENUM(リナンバー)

プログラムを修正したり新しい命令文をつけ加えたりしているうちに、行番号が不ぞろいになったり、行と行がつまりすぎて新しい命令文をつけ加えることができなくなることがあります。そんなときに行番号を、好きなようにつけ直せる命令があれば便利ですね。それをするのがRENUM命令です。さっきのプログラムを使ってちょっと確かめてみましょう。

RENUM　1，100，1 (RETURN)
LIST (RETURN)

と打ってみましょう。行番号が1から1おきに変わっていますね。

```
RENUM 1,100,1
Ok
LIST
1 A=10
2 B=A*10+3
3 PRINT B
4 END
Ok
█
```

RENUM 1，100，1というのは「行番号100を1に変えて、そこから1おきに番号をつけ直しなさい」という意味の命令なのです。

RENUM　新行番号，旧行番号，間隔

RENUMとだけ書いてあとを省略すると、始めが10で以下10ずつ増えていく行番号につけ変わります。

RENUM (RETURN)
LIST (RETURN)

```
RENUM
Ok
LIST
10 A=10
20 B=A*10+3
30 PRINT B
40 END
Ok
█
```

ちゃんと変わっていますね。

RENUM	100	……先頭の行番号を100にして、10間隔にかえる
RENUM	100,50	…行番号50を100にして、それ以後を10間隔に
RENUM	,50	……行番号50を10にしてそれ以後を10間隔に
RENUM	,50,5	…行番号50を10にしてそれ以後を5間隔に

●DELETE(デリート)

1行全部をプログラムから消してしまう方法は、行番号だけ打ち込んで RETURN とすればよいのでしたね。でも、多くの行をいっぺんに消したいときはどうすればよいのでしょうか。いちいち行番号を打ち込んで RETURN としていたのではたいへんですね。そんなときに便利なのがDELETE 命令です。

DELETE	消したい最初の行番号ー消したい最後の行番号

こうすることで消そうとする行が、プログラムから一度に消えることになります。さっきのプログラムで、ちょっとためしてみましょう。まず LIST をとってみてください。行番号10から10おきになっていますね。そこで

DELETE　10－20 RETURN
LIST RETURN

としてみて下さい。行番号10、行番号20がなくなっているのがわかりますね。

```
DELETE 10-20
Ok
LIST
30 PRINT B
40 END
Ok
█
```

DELETE　150…	行番号150を消す
DELETE　－150…	はじめから行番号150までを消す

●REM(リマーク)

REM はプログラムに説明を入れるための命令です。次のようなプログラムを作って RUN させてください。

```
10　REM テスト　プログラム　デス
20　REM A ニ 10ヲ　イレマス
30　A＝10
40　REM Aノ　ナイヨウヲ　ガメンニ
　　カキマス
50　PRINT　A
60　REM オワリデス
70　END
```

このプログラムでそれぞれの行の REM 命令より後に書いたものは、それが何であっても実行されません。REM の後には好きなことを書いてよいのです。普通はプログラムをわかりやすくするために、プログラムの説明などを書いておきます。

```
10 REM テスト プログラム デス
20 REM A ニ 10ヲ イレマス
30 A=10
40 REM Aノ ナイヨウヲ ガメンニ カキマス
50 PRINT A
60 REM オワリデス
70 END
RUN
 10
Ok
█
```

●REMの省略形

REMと書くかわりに、省略形「'」(アポストロフィ)記号を用いて、REMと同じ働きをさせることができます。たとえば上のプログラムの10行目、20行目はつぎのようになります。

```
10　'テスト　プログラム　デス
20　'A ニ 10 ヲ　イレマス
```

GOTO文で流れをかえよう
…GOTOの使い方
(GOTO,CONT)

●GOTO（ゴートゥー）

プログラムは行番号順に順序正しく実行していくと言いましたね。しかし、場合によっては順序をかえて実行させたいときがあります。こんなときに **GOTO** 命令が便利な働きをしてくれます。

次のようなプログラムを打ち込んでください。

```
10  A = 1
20  PRINT  A
30  A = A + 1
40  GOTO  20
50  END
```

GOTO は、「そのあとに続く数字の行番号の所に飛びなさい」という意味です。ですから GOTO 20 は「行番号20へ飛びなさい」という意味になりますね。

GOTO 行番号

0～65529の数値
変数や式ではエラーになる

では、プログラムを実行してみましょう。

```
RUN
 1
 2
 3
 4
```

（実際にはもっと数字が続いて表示されます。）

1つずつ増えていく数が次々に画面に書かれていき、いつまでたっても止まりませんね。考えてみれば当然なのです。ベーシックは行番号40まで次々と実行していきますが、40行の GOTO　20に出会って20行にとんでしまい、50行のENDは決して実行しないからです。

さて、それではどうすれば止まるでしょうか？

キーボードの左の方にある [CTRL] キーを押しながら上の方にある [STOP] キーを押してください。画面に

Break in 20（30, 40のときもあります。）

というメッセージが表示されて、プログラムは止まります。このメッセージは「プログラムの実行は、行番号20で止まりました」という意味です。

●CONT（コンティニュー）

ベーシックには、止めた所からもう一度実行を開始させる CONT 命令があります。

CONT [RETURN]

と打ち込んでください。[STOP] キーを押して止まった行番号の所から再び実行を始めるはずです。
実行を再開したくなければ、CONT 命令を実行する必要はありません。

```
  101
  102
  103
Break in 20
Ok
CONT
  104
```

（CONT の矢印注記）実行を再開する。

(実際にはもっと数字が続いて表示されます)

〈起こりやすいエラー〉

●Undefined line number

(アンデファインド　ライン　ナンバー)

GOTO 文の飛び先である行番号がどこにもない場合に起こります。たとえば

GOTO　1000

とプログラムの中に書いてあるのに行番号1000がないような場合です。

●Can't CONTINUE

(キャント　コンティニュー)

CONT 命令で、実行が再開できない場合に起こります。一度も RUN させていないのに、CONT命令を実行させた場合などです。

「，」と「；」の大研究
…さらにくわしいPRINT命令

PRINT 命令は、プログラムの中でよく使われる命令です。これまで何度か出ています。ここでは、PRINT 命令といっしょに使うととても便利な「，」(カンマ)と「；」(セミコロン)について勉強しましょう。

```
10  A＝1
20  B＝2
30  C＝A＋B
40  D＝A－B
50  E＝A＊B
60  F＝A／B
70  PRINT  C，D，E，F
80  END
```

とプログラムを打ち込んでください。行番号70に注目してください。

```
70  PRINT  C, D, E, F
```

C, D, E ,Fの間に「，」が入っていますね。この「，」はどんな働きをするのでしょうか？とにかく RUN させてみましょう。

```
RUN
 3              -1
↑2               .5
Ok
■
```

数値の場合は「－」などの記号がつくことがあるので、そのために1文字分あいています。

画面は図のようになります。PRINT 命令に「，」を使って変数をならべると、最初の変数の内容は始めの14文字分に、2つめの変数の内容はあとの14文字分にそれぞれ表示されます。(画面は29文字表示なので終わりの一文字分は残り、次の変数は次の行に表示されます。)次の変数も、その次の行に同じように表示されます。

今度は行番号70を次のように修正してください。

```
70  PRINT  C; D; E; F
```

「，」を「；」に変えるだけですよ。修正の方法は覚えていますね。では、さっそく RUN してみましょう。

「－」のために1文字あいています。

```
␣3␣-1␣␣2␣␣.5
Ok
■
```

後に1文字分あきます。

「；」は表示する文字や記号をぴったりとくっつけてしまうのです。変数だけでなくて「”」「”」ではさんだ文字などの場合も同じことができます。次のように打ち込んでみましょう。

PRINT　”ＡＢＣ”，”ＤＥＦ” RETURN
PRINT　”ＡＢＣ”；”ＤＥＦ” RETURN

```
PRINT "ABC","DEF"
ABC             DEF
Ok
PRINT "ABC";"DEF"
ABCDEF
Ok
█
```

文字の場合「－」のための空白と後の空白はでない。

ただし、「，」の場合には、長い文字列や小数点が入って14文字以上になると次の行に表示されます。

「；」では、文字はぴったりくっつきますが、数字は前に「－」符号のために一文字分空白があき、後にも一文字分の空白があくのでぴったりくっつくことはありません。数字を文字として扱えば、くっつけることができます。次のように打ち込んで確かめてみましょう。

PRINT　10；５；３ RETURN
PRINT　”10”；”５”；”３” RETURN

```
PRINT 10;5;3
 10  5  3
Ok
PRINT "10";"5";"3"
1053
Ok
█
```

PRINT　○，○，○，……
PRINT　○；○；○；………

PRINTを助ける三銃士 …PRINTといっしょに使う便利な命令や関数 (SPC関数, TAB関数, LOCATE, WIDTH)

PRINT命令に「,」や「;」を組み合わせて使うと、画面にいろいろな書き方で文字が表示できましたね。本機にはPRINTと組み合わせて使う、もっと便利な命令や関数がいくつかあります。

●SPC（スペース）関数

SPC関数は好きな数だけスペース（空白）を画面に書くことができます。

```
SPC（あけたい空白の数）
```

0～255の数値、数値変数、式
かっこを忘れずに

```
PRINT SPC(5); "A" ; SPC(6) ; 4 * 5
RETURN
```

と打ち込んでみましょう。SPC（5）はスペース5つぶん、SPC（6）はスペース6つぶんということがよくわかりますね。

```
Ok
PRINT SPC(5);"A";SPC(6);4*5
     A      20
Ok
```

5文字の空白　6文字の空白　「－」記号のための空白

PRINT命令とLPRINT命令の中でしか使えないので注意してください。

PRINT SPC(5) ; "A" ; SPC(6) ; 4 * 5

前後に何かがあるときは「;」ではさむこと

●TAB（タブ）関数

行の左はしから好きな数だけあけて書き始めたいときには、TAB関数を使います。

TAB（左からあけたい文字の数）

TAB関数もPRINT命令とLPRINT命令の中でしか使うことができません。
今度は次のように打ち込んでください。

PRINT TAB(5);"A";TAB(13);4*5 RETURN

```
Ok
PRINT TAB(5);"A";TAB(13);4*5
     A       20
Ok
```

5文字分　13文字分　「−」記号のための空白

TABも前後に何かあるときは「;」ではさもう

行の左はしを1個めと数えるので、TAB(5)は5文字分空白をあけるのと同じになるわけです。

●LOCATE（ロケイト）

画面にある小さな四角「■」をカーソルということはもう説明しましたね。PRINT命令は、現在のカーソルがある場所に文字を書く命令です。ですから、プログラムで自由にカーソルを動かすことができれば、好きな場所から文字を書き始めることができるわけです。
今、あなたが見ている画面は次のようになっています。縦に0～22までの23、横に0～28までの29文字を表示できるようになっています。
この後に説明するWIDTH命令を使うと画面に表示できる文字数を変えることもできます。

横→　(29文字表示の場合)
縦↓
(0，0)　(28，0)
(0，22)　(28，22)

（0，0）などの数字は表示する位置を示すもので、文字を好きなところにかくためには、この数字で位置を指定してその位置にカーソルを動かさなければなりません。
このカーソルを動かす命令がLOCATE命令です。

LOCATE　横方向の位置，縦方向の位置

数値、変数または式
画面の大きさを超えない値

画面の下には、ファンクションキーの機能が表示されていますので、実際には縦方向21文字までしか使えませんので注意してください。

LOCATE命令を使って、プログラムを作ってみましょう。

```
10  CLS
20  LOCATE  10,10
30  PRINT "A"
40  END
```

縦10文字分、横10文字分の位置に「A」が表示されますね。

←10文字分　→

10文字分

Ok

A

以上のようにSPC、TAB関数やLOCATE命令と組み合わせて使うと好きな場所に文字を書くことができます。

おおいに活用できそうですね。

PRINT命令と直接組み合わせて使うことはありませんが、画面に文字を表示するのに大切な命令を説明します。

画面に表示できる文字数を変える命令です。

●WIDTH（ウィドス）

WIDTH命令は１行に表示する文字の数を決める命令です。

WIDTH　１行に表示する文字数

↑

１～32までの数値

WIDTH　32　RETURN

と打ち込んでみましょう。

Ok

32文字分

１行に表示する文字の数をカーソルを動かして調べてみましょう。

32文字になっていることがわかりますね。

こんどは文字数を少なくしてみましょう。

WIDTH　10　RETURN

Ok

11文字分　　11文字分

左右に空白ができて、１行に10文字しか表示していませんね。

こんどはもとに戻してみましょう。

WIDTH　29　RETURN

これでもとに戻りますね。
この命令は後に出てくるSCREEN命令を使って文字を表示するもう一つの画面を表示させると、1行に入る文字の数を1～80まで変えることができますが、こちらは文法編のSCREEN命令をご覧ください。

〈起こりやすいエラー〉

●Illegal function call
　(イリーガル　ファンクション　コール)

SPCやTAB, LOCATEに使う数値が決められた範囲をこえた場合に起こります。

文字列って何？
･･･文字列の使い方

文字列というのは数字や、アルファベット、記号などをPRINT命令の中などで「”」（ダブルクォート）でくくったものです。たとえば

”ABCDEFG” や ”わたしは　コンピュータです”

などは文字列です。「”」の中に使える文字は、キーボードから打てる文字なら、どのようなものを使ってもかまいません。ただし、「”」だけは使うことができません。では文字列を画面に書いてみましょう。

```
PRINT　”わたしは　コンピュータです” RETURN
```

とすればよいのでしたね。
文字列を使うことによってコンピュータが人間にわかる言葉を作り出すのです。

●文字変数を使おう

いままで使ってきた変数は数を表すものでした。変数には、数を表すものばかりでなく、文字列を表すものもあるのです。それを文字変数といいます。
次のように打ち込んでください。

```
A$ =”わたしは　コンピュータです” RETURN
PRINT　A$ RETURN
```

PRINT”わたしは　コンピュータです”としたときと同じ結果になりましたね。ここで使った「A$」が文字変数で、「わたしは　コンピュータです」という文字列がはいっているわけです。

文字変数の名前の最後に「$」（ドル）記号をつけます。この記号をつけることで、これまでにでてきた変数（数値変数といいます）と区別しているわけです。名前のつけ方は27ページの「変数名の3つの規則」をよく読んでください。

●文字列のつなぎあわせ

文字列や文字変数は数ではないので、たし算、ひき算、かけ算、わり算などの四則演算はできません。ですからPRINT"123"+25やPRINT"2"*3+2などという計算を行なうとエラーになります。
そのかわり、つないだり、切ったり、抜きだしたりすることができます。ここでは、文字列のつなぎ方について説明しましょう。

```
PRINT "モジレツ ヲ "+"ツナイデ
ミヨウ" RETURN
```

と打ち込んでください。「モジレツ　ヲ　ツナイデ　ミヨウ」と1つにつながって表示されますね。今度は

```
10  A$="モジレツ ヲ "
20  B$="ツナイデ ミヨウ"
30  C$=A$+B$
40  PRINT C$
50  END
```

として**RUN**させてください。同じ結果になりましたね。
このようにいくつかの文字列や文字変数を「+」によってつなぐことができるのです。これを文字式といいます。ここで使った「+」はたし算の「+」とは意味が違うことに注意してください。

〈起こりやすいエラー〉

●Type mismatch（タイプ　ミスマッチ）
数値と文字列を混同して使った場合などに起こります。
たとえば

```
PRINT "123" +123
```

"123"は文字列ですが123は数値です。また

```
PRINT A$+B
```

A$は文字変数ですがBは数値変数です。いずれの場合もエラーになります。

こちらINPUT応答せよ

コンピュータに変数の値を知らせるために、いままでは、

```
10  A=2
20  B=5
  ⋮
```

のように、始めに数値を決めてしまう方法をとってきました。変数の値を変えたい場合には、プログラムを修正しなければならないのです。そこで、必要に応じてキーボードから変数の値を変えられるINPUT命令の登場です。

●INPUT（インプット）

円の面積を求める次のプログラムを打ち込んでRUNさせてみましょう。

```
10  INPUT  R
20  S=3.14159*R*R
30  PRINT  S
40  GOTO  10
50  END
```

見なれない表示が出てきましたね。「？」(クエスチョンマーク）は、コンピュータがあなたに、「変数Rの値は何にしますか？」ときいてきているのです。

たとえば、

10 RETURN

と打ち込んでみましょう。コンピュータは，3.14159×10×10を計算して、

314.159

と答えを出すはずです。いろいろな数を入力してためしてみてください。

```
RUN
? ■  ←カーソル
```

```
RUN
? 10
 314.159
? 1
 3.14159
? 2.2
 15.2052956
?  ← ここで CTRL + STOP キーを押した
Break in 10
Ok
■
```

●たくさんの変数への入力

さきほどのプログラムでは、値を入力する変数はRただ1つでしたが、変数がたくさんある場合はどうしたらよいでしょうか。
次のプログラムは、一度に入力された4つの数の平均を出すプログラムです。

```
10  INPUT   A, B, C, D
20  PRINT   (A+B+C+D)/4
30  GOTO   10
40  END
```

```
RUN
? 10,20,30,40
 25
? 1,2,3,4
 2.5
? ←――――――― ここで CTRL + STOP
Break in 10        キーを押した。
Ok
█
```

RUNさせて「?」がでてきたら、数字を打ち込みます。
たとえば、

10, 20, 30, 40 RETURN

と入力してください。打ちまちがえたときは、RETURN を押す前ならば、カーソルキーや DEL 、INS 、BS キーで修正することができます。

たくさんの変数にINPUT命令で値を入れる場合は、「，」をはさんで入れることを覚えておいてください。

INPUT　変数, 変数, ……, 変数

前ページのプログラムをRUNさせて、次のように数値を入力してみましょう。

15，20，30 RETURN

入力する変数の値が1つ少ないですね。

コンピュータは、「？」(疑問符)を2つ表示します。
これは入力される変数の個数が足りませんという意味ですから、つづけて、

35 RETURN

とします。

INPUTで書いた変数の個数が入力した値の個数よりも多いときは「？？」となります．
このときはつづけて入力します
反対に，入力した値の個数の方が多いときは「Extra ignored」となり，入力された値のうち，INPUTで書いた変数の個数分だけ読みとります

●文字変数にも入力できるよ

今度は、

```
10  INPUT   A$, B$
20  PRINT   B$; A$
30  GOTO    10
40  END
```

としてRUNしてみましょう。

```
RUN
? AAA,BBB
BBBAAA
? ﾊﾟｿｺﾝﾃﾞｽ,ﾜﾀｼﾊ
ﾜﾀｼﾊﾊﾟｿｺﾝﾃﾞｽ
?
Break in 10
Ok
█
```

INPUT命令を使って文字変数に値を入れるときには，「"」で囲む必要はない。

ここで CTRL ＋ STOP キーを押した。

文字変数にも値（文字列）を入力できることがわかりますね。
2つ以上を一度に入力するときに「，」を間に入れることは、数値を入力する場合と同じです。

●INPUT命令で画面表示

INPUT 命令を使った場合、画面に

? ■←カーソル

と出てきますが、これでは、数値を要求しているのか文字列を要求しているのか、あるいは入力する個数はいくつなのか、まったくわかりませんね。
そこで、INPUT 命令の後に、「"」でかこんだ文字列を置くと、その文字列を、「?」といっしょに表示してくれます。

INPUT "文字列"；変数,変数,……,変数

それでは、さっき作ったプログラムの行番号10を次のように書き直してRUN してみましょう。

```
10  INPUT "モジヲ  2ツ";A$,B$
```

```
RUN
ﾓｼﾞｦ 2ﾂ? ABC,DEF
DEFABC
ﾓｼﾞｦ 2ﾂ? ﾊﾟｿｺﾝﾃﾞｽ,ﾜﾀｼﾊ
ﾜﾀｼﾊﾊﾟｿｺﾝﾃﾞｽ
ﾓｼﾞｦ 2ﾂ?
Break in 10
Ok
```

← CTRL + STOP キー

このほうが、わかりやすいですね。

〈起こりやすいエラー〉

●Redo from start（リドゥー フロム スタート）

数値変数へ入力する INPUT 命令なのに、実際に入力されたものが、数値でなかった場合に起こります。
たとえば、

```
10  INPUT  A
```

というプログラムで、数値を入力するべきなのに、

? ABC RETURN

などと、文字を入力してしまった場合です。

●Extra ignored（エクストラ イグノアード）

INPUT で書かれた変数の個数よりも、キーで入力された値の個数が多いときに起ります。入力された値は、変数の個数の分だけ、先頭から読みこまれ、残りは無視されます。

```
10  INPUT  A, B, C
```

というプログラムで、

? 10, 20, 40, 50, 60 RETURN

とすると、50, 60は無視されます。

数の料理の専門家

・・・ 数値関数

(ABS関数,SGN関数,INT関数)

ここでは、数値を料理する関数を説明しましょう。

●ABS(アブソリュート)関数

絶対値という言葉を聞いたことがありますか？　正の数はそのままに、負の数は「−」(マイナス)を取ったものを値とするのが絶対値です。たとえば次のように使います。

5の絶対値は5
−3の絶対値は3
0の絶対値は0

この絶対値を与えるのがABS関数です。

ABS(数値か変数か式)

かっこを忘れずに

次のプログラムをRUNさせてみましょう。

```
10  INPUT  X
20  PRINT  ABS  (X)
30  GOTO  10
```

正の数や、負の数をキーボードから入力してください。「−」がなくなって画面に表示されますね。

```
RUN
? 10.5
 10.5
? -10.5
 10.5
? 0
 0
?
Break in 10
Ok
```

CTRL + STOP キー

●SGN(サイン)関数

正の数ならば1，負の数ならば−1、0ならば0というように数の符号を与えます。

```
SGN(数値か変数か式)
```

かっこを忘れないこと

使い方は、ABS関数と同じです。さっきのプログラムの行番号20を次のように変えて、RUNしてみましょう。

```
20  PRINT  SGN (X)
```

```
RUN
? -20
-1
? 10.23
 1
? 0
 0
?
Break in 10
Ok
```

CTRL ＋ STOP キー

●INT(インテジャー)関数

INT 関数は、小数点以下の部分を切り捨てて、整数にする命令です。

```
INT(数値か変数か式)
```

今度は、行番号20を次のようにして、RUNさせてみましょう。

```
20  PRINT  INT (X)
```

```
RUN
? 1.5
 1
? 0.01
 0
? -2.1
-3
? -0.01
-1
? 0
 0
?
Break in 10
Ok
```

CTRL ＋ STOP キー

INT関数では、与えられた数値の小数点以下の部分を、負の方向に切り捨てます。たとえば、4.25のときに4、−5.3のときには−6となりますから、INT関数では、その結果が、与えられた数値よりも、必ず小さくなります。

●ちょっと関数のお話

ABS、SGN、INTなどの数を料理する専門家を説明してきました。これらは関数と呼ばれています。関数とは「ある数値が与えられ、それになんらかの処理をした結果を値として持つもの」をいいます。たとえば、ABS(－2)というのは、－2という数が与えられて、それに絶対値をとるという処理をした結果の、2という値を持っています。SGN、INTも同じことが言えますね。

関数は、それ自身、値を持つので、式の中に書くこともできるし、PRINT命令の右側に書くこともできるのです。それで、

```
PRINT  INT(5.6)*SGN(-6)+3
```

などをちゃんと計算してくれますね。

また、後で説明しますがLEFT$、RIGHT$、MID$なども似たような性質を持っています。(詳しいことは74ページからお読みください。) たとえば、LEFT$("ABCD", 2)というのは、"ABCD"という文字列と2という数が与えられて、「与えられた文字列の左から、与えられた数字の分だけ取り出す」という処理をした結果、"AB" という値を持っています。

また、

```
PRINT  LEFT$ ("1234", 2)+"56"
```

などと文字式の中に使うことができます。RIGHT$、MID$も同じことが言えます。

LEFT$、RIGHT$、MID$も関数なのです。ただし、値として持つのが文字列なので、これらは文字関数と呼ばれています。それに対して、ABS、SGN、INTなどは数値関数と呼ばれています。

●その他の関数

本機はこの他にも、たくさんの関数を持っています。

詳しいことはベーシック文法編をお読みください。

Ⅳ ベーシックは便利だ

ベーシックの命令やはたらきには、便利なものがたくさんあります。そのなかで、よく使われる同じ仕事を繰り返す命令やコンピュータに判断をさせる命令、そして作ったプログラムに誤りがあるときに、それを修正する方法などを紹介します。
雑誌などに掲載されているプログラムをご覧になると必ず出てくる命令です。
最後には少し長いゲームのプログラムを載せてあります。ぜひ入力してみてください。

こんなこともできるよ

判断はIF～THENにおまかせ

(IF～THEN (ELSE))

本機をますますコンピュータらしく使う命令を紹介しましょう。条件判断をするIF～THEN命令です。

●IF～THEN（イフ～ゼン）

IF～THEN命令は「もし～だったら…」という条件判断を行う場合に使います。

```
40  IF△△△THEN□□□
40  ■■■
```

「もし△△△の条件が満たされるのならば、□□□を実行すること。満たされないならば□□□は無視して次の行の■■■を実行しなさい」というのがIF～THEN命令です。

次のプログラムをRUNさせてみましょう。

```
10  INPUT X
20  IF X > 10 THEN PRINT
     " 10 ヨリオオキイ" : GOTO 10
30  PRINT " 10  イカデス "
40  GOTO 10
50  END
```

「？」が出てカーソルが出ますので、何か数字をいれて RETURN としてください。いれた数が10より大きいときは、

```
10  ヨリオオキイ
```

10以下のときは、

```
10  イカデス
```

と答えてくれますね。

```
RUN
? 2
10 イカデス
? 46
10 ヨリオオキイ
? 777
10 ヨリオオキイ
? -98
10 イカデス
? 4.873
10 イカデス
? 0.987
10 イカデス
? -100.45
10 イカデス
? 46764878
10 ヨリオオキイ
?
Break in 10
Ok
■
```

CTRL + STOP キー

プログラムの行番号20にあるX＞10のように２つの値を比較する式を条件式と呼びます。

IF 条件式 THEN 命令：命令：…

「：」でつないでTHENの後に命令を続けて書くことができます。
条件が満たされた場合にのみこの後の命令が実行されます。

記号	条件式の書きかた	書きかた
＝	AとBは等しい	A＝B
＞	AはBより大きい	A＞B
＜	AはBより小さい	A＜B
＜＞または＞＜	AとBは等しくない	A＜＞B
＞＝または＝＞	AはB以上	A＞＝B
＜＝または＝＜	AはB以下	A＜＝B

また、THENのすぐ後にGOTO命令を書くこともできます。

IF 条件式 THEN GOTO 行番号

このときは、GOTO命令か、THENのどちらかをはぶくことができます。

IF 条件式 THEN 行番号
IF 条件式 GOTO 行番号

と、どちらでもかまいません。

●ELSE（エルス）

IF～THEN命令にELSEをつけ加えてIF～THEN～ELSEという命令にすることができます。

```
IF△△△THEN□□□ELSE■■■
```

「もし△△△の条件が満たされるならば□□□を実行し、満たされないなら□□□を飛ばして■■■を実行しなさい」という命令になります。
条件式や命令の書き方はIF～THEN命令と同じです。
さきほどのプログラムをIF～THEN～ELSEを使って書き換えてみましょう。

```
10  INPUT X
20  IF X > 10 THEN PRINT "10
    ヨリオオキイ" ELSE PRINT"
    10 イカデス"
30  GOTO 10
40  END
```

この方がプログラムが簡単になりますね。
RUNの結果はさっきと同じですね。
また、

```
IF 条件式 THEN 命令 ELSE GOTO 行番号
```

のときはGOTO命令をはぶいて、

```
IF 条件式 THEN 命命 ELSE 行番号
```

と書くことができます。

●AND, OR（アンド，オア）

複雑な条件式を使う時に、このAND，ORが役に立ちます。
たとえば、「Xが10以上であって、しかも20以下」という条件式はANDを使って、

```
X>=10 AND X<=20
```

と簡単に書くことができます。
また、「Xが負の数か、または100以上」という条件式はORを使って、

```
X<0 OR X>=100
```

とすれば簡単に書くことができます。

```
条件式 1 AND 条件式 2
```

式1そして、しかも式2

```
条件式 1 OR 条件式2
```

式1 かまたは式2

身長と体重の差が100より大きく、110よりも小さいなら「ベストコンディション」、そうでないなら「フトリスギ　マタハヤセスギデス！」と出力するプログラムを作ってみましょう。

```
10  INPUT"シンチョウcm,タイジュウkg"
    ;X , Y
20  Z=X-Y
30  IF Z>100 AND Z<110
    THEN PRINT"ベストコンディション"
    ELSE PRINT" フトリスギ マタハ
    ヤセスギデス"
40  GOTO 10
50  END
```

```
RUN
シンチョウcm,タイジュウkg? 173,60
フトリスギ  マタハ ヤセスギデス
シンチョウcm,タイジュウkg? 173,65
ベストコンディション
シンチョウcm,タイジュウkg? 153,48
ベストコンディション
シンチョウcm,タイジュウkg?
Break in 10
Ok
█
```

CTRL + STOP キー

くり返しに便利な命令
・・・FOR～NEXTの使い方

プログラムの中で、同じことを何度もくり返したい場合がよくあります。これまでは GOTO 文を使ってくり返しをして、CTRL キーを押しながら STOP キーを押すことで終わらせていましたが、一定の回数だけくり返したら自動的に終わらせるというようなときに便利なのが、FOR～NEXT 命令です。

```
RUN
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
Ok
```

●FOR～NEXT
（フォー～ネクスト）

まず、1から10までの数を画面に書くプログラムを作って RUN させてみましょう。

```
10  FOR  N＝1  TO  10
20  PRINT  N
30  NEXT  N
40  END
```

10　Nを1から10まで1ずつふやしながら、次のことをくり返しなさい。
20　Nのなかみを画面に書きなさい。
30　ここまでをくり返しなさい。
40　終りです。

FOR～NEXT命令は、FOR～とNEXT～の間に囲まれた部分（このプログラムでは、20 PRINT Nがその部分です）をある決められた回数だけくり返す命令です。くり返しの数を覚えている変数（**制御変数**ともいいます）を使えるので、とても便利ですね。

FOR 変数＝初めの値 TO 終りの値

- 変数 ← くり返しの数を覚えておくための変数
- ＝ ← 「＝」イコールです。
- 初めの値・終りの値 ← 変数、式でもよい。

くり返す命令(いくつ書いてもよい)

NEXT 変数

- 変数 ← FORの次に書いた変数と同じものを使うこと。

この変数の変化を1，2，3と1ずつ大きくするのではなく、たとえば0.5づつ大きくするとか、あるいは2ずつ小さくすることもできます。このときの変化幅を**STEP** という命令で指定します。

```
FOR  I＝1   TO  10 [STEP 0.5]
FOR  A＝10  TO −10 [STEP −2]
```

（STEPの値は）変数、式でよい。

それでは、さきほどのプログラムを，2，4，6，…10と書くプログラムに修正してみましょう。

```
10 FOR N＝[2] TO 10 [STEP 2] RETURN
```

四角で囲んだところだけを修正するのですよ。できたら、**RUN** させてみましょう。

```
RUN
 2
 4
 6
 8
 10
Ok
█
```

●もっと複雑なFOR～NEXTにチャレンジ

FOR～TO～NEXT 命令は、2重3重に重ねて使うことができます。しかし、必ず守らなければならない規則があります。

(1) 内側の FOR～NEXT と外側の FOR～NEXT 命令が交差してはいけません。
(2) FOR～NEXT 命令の内側から GOTO 命令や IF～THEN 命令（60ページ）などで外側に飛び出すことはできますが、FOR～NEXT 命令の外側から内側に飛び込んではいけません。

```
┌─ FOR  A=1  TO  10
│┌─ FOR  B=2  TO  5
││┌─ FOR  C=3  TO  4
│││  くり返す命令
││└─ NEXT  C
│└─ NEXT  B
└─ NEXT  A
```

●よい例

```
FOR  A=1  TO  10
FOR  B=2  TO  5
FOR  C=3  TO  4
 くり返す命令
NEXT  A
NEXT  B
NEXT  C
```

●悪い例

FOR～NEXT は交差してはいけない。

```
 ┌─ 50  GOTO  200
 │┌120  FOR  I = 0  TO  N
 └→200  ………………………………
  └270  NEXT  I
```

●悪い例

FOR～NEXT への飛び込みはいけない。

それでは、2重の FOR～NEXT 命令を使って、九九の計算表を作るプログラムを書いてみましょう。

```
10  CLS
20  FOR  X=1  TO  9
30  FOR  Y=1  TO  9
40  LOCATE  3*X-3, 2*Y:
    PRINT  X*Y
50  NEXT  Y
60  NEXT  X
70  END
```

変数Xの FOR～NEXT 命令のくり返しの中に、変数Yの FOR～NEXT 命令のくり返しがはさまれています。変数Xと変数Yはそれぞれ1から9まで変化しますが、最初は変数Xに1がはいり、Y＝1, 2, …, …, 9とくり返します。変数Yが9までいくと、内側のくり返しは終わりになり、今度はX＝2となり同じことをくり返すわけです。
では、さっそく RUN させてみましょう。

```
RUN
 1   2   3   4   5   6   7   8   9
 2   4   6   8  10  12  14  16  18
 3   6   9  12  15  18  21  24  27
 4   8  12  16  20  24  28  32  36
 5  10  15  20  25  30  35  40  45
 6  12  18  24  30  36  42  48  54
 7  14  21  28  35  42  49  56  63
 8  16  24  32  40  48  56  64  72
 9  18  27  36  45  54  63  72  81
Ok
█
```

きれいな九九の表ができ上りました。

〈起こりやすいエラー〉

●NEXT without FOR
（ネクスト　ウィズアウト　フォー）

FORを実行していないのにNEXTを実行したときに起こります。FOR～NEXT命令の外側から内側に飛び込んだときや、FOR の制御変数とNEXTの制御変数が違っている場合などに起こります。

でたらめな数をRND関数で作ろう

ここまでコンピュータを操作してきますと「コンピュータは規則正しいことはできるが、でたらめなことはできない」あなたはそう思っていませんでしたか。ところが違うのです。コンピュータにはでたらめな数を作り出す**RND**関数があります。

●RND(ランダム)関数

RND関数は、0より大きい1未満のでたらめな数(乱数といいます)を作る関数です。
普通は、

```
RND (1)
```

と書きますが、「1」のところは正の数なら何でもかまいません。すべて同じ結果になります。

RND関数を使ってコンピュータ・サイコロを作ってみましょう。
サイコロのワク組の線はグラフィック文字ですから、GRAPH キーを押しながら書いてください。

```
10  X=INT(RND(1)*6)+1
20  PRINT  "┌───┐"
30  PRINT  "│␣␣␣│"
40  PRINT  "│";X;"│"
50  PRINT  "│␣␣␣│"
60  PRINT  "└───┘"
70  INPUT  "モウイチド";A$
80  GOTO  10
90  END
```

注)「␣」は一文字分の空白(スペース)を意味します。

行番号10の

```
INT  (RND(1)*6)+1
```

は次のような計算をしているのです。

RND(1)	0より大きく1より小さい
RND(1)*6	0より大きく6より小さい
INT(RND(1)*6)	0,1,2,3,4,5のどれか
INT(RND(1)*6)+1	1,2,3,4,5,6のどれか

となるので、ここでサイコロの目を計算していることがわかります。サイコロの目のように、結果が不規則なものを作り出すには**RND**関数を使うことが便利ということがわかりますね。
それではさっそくRUNしてみましょう。

モウイチド?

と表示されたら適当な文字や数値を入れて RETURN キーを押してください。

```
RUN
 5

モウイチト゛？  Y
 3

モウイチト゛？  Y
 6

モウイチト゛？
Break in 70
Ok
■
```

適当(てきとう)なキーを押(お)して RETURN としてください。

CTRL + STOP キーを押(お)した

りっぱなコンピュータ・サイコロのできあがりです。

同じような処理には サブルーチンを使おう！ …GOSUB～RETURNの使い方

プログラムを作っていくと同じ処理があちこちにでてくることがよくあります。同じ命令を２度も３度も書くのは大変手数のかかるものです。例えば、同じ文章を何回もプログラムの中で表示させることがよくあります。
こんなときに、プログラムの共通した部分を独立したプログラムにして、必要なときにこのプログラムを使うことができます。この独立したプログラムをサブルーチンといいます。
このサブルーチンを使うための命令がGOSUB～RETURNなのです。

●GOSUB～RETURN（ゴーサブ～リターン）

GOSUBはサブルーチンに飛んで行きなさいという命令です。

GOSUB 行番号

とプログラムの中に書いておくと「行番号」（普通サブルーチンの先頭です。）へ飛んで、そこで処理を行ないます。ここまではGOTO命令と同じですね。ただ違うのは、飛び先のサブルーチンで命令を実行していくうちにRETURN命令に出会うと、もとのGOSUB命令に戻ってきて次の命令を実行するという点です。

GOSUB～RETURN命令を使って計算練習のプログラムを作ってみましょう。RND、INT関数は説明しましたね。

```
10  K=0
20  FOR N=1 TO 10
30  X=INT (RND (1) *100)
40  Y=INT (RND (1) *100)
50  Z=X+Y
60  PRINT X ; " + " ; Y
70  INPUT " こたえは" ; W
80  IF Z=W THEN GOSUB 120
90  NEXT N
100 IF K>8 THEN GOSUB 120
110 END
120 PRINT " よく できました！"
130 PLAY "V13L8CDFCDFCDL2F"
140 K=K+1
150 RETURN
```

行番号120～150がサブルーチンです。END命令の後になっていますが、サブルーチンはプログラムの中のどこに書いてもかまいません。ただEND命令より前に書くとGOSUB命令で飛び込んだのではないのにRETURN命令を実行しようとしてエラーになることがありますので注意してください。サブルーチンの仕事は入力した答えが正解だったとき、全部で10問の計算練習のうち８問以上が正解のときに「よく　できました。」という表示をします。行番号80と100からサブルーチンに飛び込んでいるのがわかりますね。

行番号130は音楽を演奏する命令です。くわしくは**音だって出せるんだ**で説明しますので、ここではプログラムのとおり打ち込んでください。

```
 59 + 10
こたえは?
```

RUNしたあとの画面です。

サブルーチンを使うとプログラムが整理され、見やすくなります。コンピュータではよく使われる大切な考えかたですから、機会があればいろいろなプログラムを見てよく理解してください。

プログラム

```
GOSUB  120
  |
GOSUB  120
  |
END
```

120行から サブルーチンに すればいいんだね！

サブルーチン

```
120----------
  |
RETURN
```

●ON GOSUB（オン ゴーサブ）

GOSUB命令をさらに便利にした命令があります。例えば、いくつかの項目のなかから一つをえらんで処理するとき、IF〜THENを使って次のようにプログラムを作ることになりますね。

```
10  INPUT ”すきないろは  1あか  2あお
    3しろ”；X
20  IF X=1 THEN GOSUB 70
30  IF X=2 THEN GOSUB 90
40  IF X=3 THEN GOSUB 110
50  IF X<1 OR X>3 THEN 10
60  END
70  PRINT ”じょうねつてきなひと”
80  RETURN
90  PRINT ”れいせいなひと”
100 RETURN
110 PRINT ”けっぺきなひと”
120 RETURN
```

こんなときに便利なのがON GOSUB命令なのです。ON GOSUB命令を使うと、プログラムは次のようになります。

```
10  INPUT ”すきないろは  1あか
    2あお  3しろ”；X
20  ON X GOSUB 70 , 90 , 110
50  IF X<1 OR X>3 THEN 10
60  END
70  PRINT ”じょうねつてきなひと”
80  RETURN
90  PRINT ”れいせいなひと”
100 RETURN
110 PRINT ”けっぺきなひと”
120 RETURN
```

行番号20から40までが、1行にまとまってしまいましたね。

ON GOSUB命令は、ONの後に書れている変数の値によって飛び先のサブルーチンを指定することができる命令です。

```
ON 変数か式 GOSUB 行番号, 行番号, …
```

この値が1のとき

2のとき

⋮

もちろんGOSUB命令と同じように、RETURN命令に出会うと戻ってきて、次の行の命令を実行することには変わりありません。

注意しなければいけないのは、ONの後に書かれた変数や式の値が、GOSUBの後に書かれた飛び先の行番号の個数よりも多いときは、ON〜GOSUB命令は無視されて次の行の命令が実行されます。

変数や式の値が負になると、エラーになってしまいます。

●ON～GOTO（オン～ゴートゥ）

GOSUBの変わりにGOTOを使うこともできます。この場合には、飛び先の行番号の命令を実行します。ON～GOSUBと同じ使い方をします。

ON 変数か式 GOTO 行番号，行番号，…

ONの後に書かている変数や式の値によってGOTO命令の飛び先を指定することができる命令です。
ONの後に書かれた変数や式の値が０になったり、GOTOの後に書かれた行番号の個数よりも大きいときには、ON～GOSUBと同じように無視されたり、エラーが出たりします。

<起こりやすいエラー>

●RETURN without GOSUB
（リターン ウィズアウト ゴーサブ）

GOSUB命令を実行していないのに、RETURN命令に出合ったときに起こります。

●Undefined line number
（アンデファインド ライン ナンバー）

GOSUB,GOTO命令の飛び先がないない場合に起こります。

●Illegal function call
（イリーガル ファンクション コール）

ONの後に書かれた変数または式の値が負になった場合に起こります。

文字列料理の３大道具 …文字列を抜き出そう
(LEFT$関数,RIGHT$関数,MID$関数,LEN関数)

文字列のつなぎかたはわかりましたね。ここでは文字列を切ったり、抜き出したりするときに使う便利な道具について説明しましょう。

●LEFT＄(レフトドル)関数

文字列や、文字変数の左から何文字かを抜き出すときに使います。

かっこも忘れないこと。
文字の式でもよい。
「，」(カンマ)を忘れずに
数値変数や式でもよい。

LEFT$（文字列か文字変数，抜き出す文字数）

次のプログラムを打ち込んでください。

```
10  FOR  I＝1  TO  10
20  PRINT  LEFT$（"1 2 3 4 5 6 7 8
    9 0",I）
30  NEXT  I
40  END
```

最初は左から１文字（つまり「1」)、次に２文字（「12」)、３文字…と、10文字まで"1 2 3 4 5 6 7 8 9 0"を左から抜き出していくプログラムです。さっそくRUNさせてみましょう。

```
RUN
1
12
123
1234
12345
123456
1234567
12345678
123456789
1234567890
Ok
█
```

うまくいきましたね。

●RIGHT$(ライトドル)関数

文字列や、文字変数の右から何文字かを抜き出すときに使います。

RIGHT$(文字列か文字変数，抜き出す文字数)

- 文字の式でもよい。
- かっこ
- 「，」を忘れずに。
- 数値変数や式でもよい。

同じように次のプログラムを RUN させてみましょう。

```
10  FOR  I=1  TO  10
20  PRINT  RIGHT$("1234567890", I)
30  NEXT  I
40  END
```

```
RUN
0
90
890
7890
67890
567890
4567890
34567890
234567890
1234567890
Ok
█
```

●MID$(ミドルドル)関数

文字列や文字変数の、好きな場所から好きな文字数だけ抜き出すときに使います。

MID$(文字列か文字変数，抜きとるはじめの場所，抜き出す文字数)

- 「，」を忘れずに
- 左からかぞえて何番目か。
- 数値変数や式でもよい。
- 文字式でもよい。
- かっこも忘れずに。

```
10  FOR  I=1  TO  10
20  PRINT  MID$("1234567890", I, 11-I)
30  NEXT  I
40  END
```

```
RUN
1234567890
234567890
34567890
4567890
567890
67890
7890
890
90
0
Ok
█
```

どうですか？ 左と右が反対になっただけで同じようなことが起きているのがわかりますね。

●LEN（レングス）関数

最後に今までのRIGHT$、LEFT$、MID$ とはちょっと違った働きをもつ関数を紹介しましょう。
LEN関数は文字列の長さを与えるものです。

PRINT LEN("ABCDEF") RETURN

としてみましょう。
答えは6となりますね。

LEN（文字列）

それでは次のようにプログラムして、RUN させてみましょう。

```
10  A$="1234567890"
20  FOR  I=1  TO  9
30  PRINT  MID$ (A$,LEN (A$)
-I, 2)
40  NEXT  I
50  END
```

```
RUN
90
89
78
67
56
45
34
23
12
Ok
█
```

「1234567890」を右から順番に2文字ずつ表示していくのがわかりますね。

〈起こりやすいエラー〉

●Type mismatch
（タイプ　ミスマッチ）

文字と数値を使い間違えたときに起こります。たとえば

MID$（A，2，3）

など、文字変数を入れるべきところに、数値変数（この場合A）を入れた場合です。

ゲームをするならINKEY$関数が便利

前に勉強したINPUT命令は、最後に RETURN としないと入力を受けつけてくれませんでしたね。また、押したキーの文字を画面に書いてしまうので、インベーダーゲームのような、画面を使うゲームには使うことができません。そこで登場するのがINKEY$関数です。

何か、キーを押してみて下さい。キーを押していれば画面に、押したキーの文字を表示します。キーを押していなければ、画面に何も書きませんね。つまり、INKEY$関数というのは、そのとき押されていたキーの文字を与える命令なのです。

●INKEY$(インキードル)関数

まず、次のプログラムをRUNさせてみましょう。

```
10  A$=INKEY$
20  PRINT  A$:GOTO  10
30  END
```

```
RUN
A
A
A

!

1
1
Break in 10 (20の場合もあります。)
```

CTRL ＋ STOP キー

INKEY$

文字を与えることに注意してね
A=INKEY$では
Type mismatch エラーだよ.
A$=INKEY$としてね

INKEY$関数は、INPUT命令のように、キー入力があるまで実行を中断してくれません。キーが押されていなくても、さっさと次の命令に移ってしまいますから、このプログラムのようにGOTO命令などを使ってくり返しINKEY$関数を実行させてください。
また、INKEY$関数は、キーを押してもその文字を画面には表示しないので注意してください。画面に表示させたい場合は、このプログラムのようにPRINT命令を実行させなければなりません。

K キー →
U キー ↑
M キー ↓
← H キー
Break in 20
CTRL + STOP キー

●INKEY$をゲームに使おう

ちょっとゲームの基礎の基礎を勉強してみましょう。押したキーによって前後左右に絵を動かすプログラムです。

```
10  X=10:Y=10:CLS
20  A$=INKEY$
30  IF  A$="U"  THEN
    Y=Y-1:GOTO  70
40  IF  A$="H"  THEN
    X=X-1:GOTO  70
50  IF  A$="M"  THEN
    Y=Y+1:GOTO  70
60  IF  A$="K"  THEN
    X=X+1
70  LOCATE  X, Y:PRINT
    "♥"
80  GOTO  20
90  END
```

(「♥」はグラフィック文字です)

さあ、RUNさせてみましょう。Uを押すと「♥」マークが上に、Hを押すと左に、Mだと下に、Kを押すと右に動きますね。

ベーシックで作られたゲームはたくさんありますが、「キーを押すと絵が動く」という部分は、基本的にはこのプログラムと同じものです。COLOR命令を使ったり、もっと複雑な絵を描かせたりすれば、楽しいゲームができ上がります。

〈起こりやすいエラー〉

● Type mismatch(タイプ ミスマッチ)
数値変数と文字変数の使い間違いなどで起こります。

「：」を使ってプログラムを短く
…マルチ ステートメントの使い方

今までは、1つの行に1つの命令しか書きませんでした。ベーシックでは「：」(コロン)で区切ることによって、1行にいくつでも命令を書くことができます。たとえば次のように書くことができます。

```
10 A=5:B=10:C=20
20 PRINT A+B+C:PRINT A*B*C:PR
INT A/B/C:END
RUN
 35
 1000
 .025
Ok
■
```

```
10 A=5
20 B=10
30 C=20
40 PRINT A+B+C
50 PRINT A*B*C
60 PRINT A/B/C
70 END
RUN
 35
 1000
 .025
Ok
■
```

書きかたは違うけれど、
実は同じプログラムです。

このように、「：」を使って1行にいくつもの命令を書いたものを、マルチステートメントと言います。長いプログラムを書くときなど、マルチステートメントにすると行番号を書かないで済むので、プログラムが短くなり、実行速度が多少速くなるなどの利点があります。しかし、1行にあまりたくさんつめ込むと、プログラムが見にくくなるので、あまりマルチステートメントを多用するのはやめましょう。

デバッグのお話

ベーシックにはだいぶなれましたか？ 命令を理解するとともにエラーのおこる回数も増えてきますね。エラーの起こる原因を**バグ**（虫という意味です）と呼びます。ここでは、バグをなくす方法 **デバッグ**（「虫取り」という意味になります)、について説明します。まず次のプログラムを打ち込んでみましょう。これは、簡単なルーレット・ゲームのプログラムです。

```
10 DIM C$(2),X$(2)
20 C$(0)="♠":C$(1)="♥":C$(2)="◆"
30 K=1000
40 CLS
50 IF K<=0 THEN 300
60 LOCATE 7,2
70 PRINT "モチテン␣:␣"; K
80 LOCATE 7,4
90 INPUT "カケテン␣:"; L
100 IF L>K THEN BEEP : GOTO 60
110 FOR I=0 TO 2
120 X$(I)=C$(RND(1)*3)
130 NEXT I
140 LOCATE 11,10
150 FOR I=0 TO 2 : PRINT X$(I); : NEXT I
160 IF INKEY$ = "" THEN 110
170 LOCATE 6,16
180 PRINT"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"
190 LOCATE 6,16
200 P=1
210 IF X$(0)=X$(1) THEN P=P+1
220 IF X$(1)=X$(2) THEN P=P+1
230 IF X$(0)=X$(2) THEN P=P+1
240 IF P=1 THEN 270
250 IF P=2 THEN 280
260 IF P>=3 THEN 290
270 PRINT "ハズレ␣マイナス␣␣";4*
    L:K=K-4*L:GOTO 50
280 PRINT "アタリ!プラス␣␣␣";L:
    K=K+L:GOTO 50
290 PRINT "オオアタリ!!プラス";5*L:
    K=K+5*L:GOTO 50
300 FOR I=1 TO 10:BEEP:NEXT I
310 CLS:LOCATE 9,10
320 PRINT "ハサンデス!!":END
```

注) 「␣」は、一文字分の空白(スペース)を意味します。

●ルーレット・ゲーム

今までのプログラムよりちょっと長くなっています。うまく打ち込めましたか？　打ち込んだらさっそくRUN してみましょう。

モチテン：1000
カケテン：？■

と画面に出てきましたね。「あなたは今1000点持っています。何点かけますか？」とコンピュータが聞いているのです。持ち点以内の数字を打ち込んで RETURN としてください。「♠」、「♥」、「◆」の記号が、画面の中央に、めまぐるしく移り変わりながら表示されていますね。適当なところでスペースキーを押すと表示がそこでストップします。そこで同じ記号が2つ出れば「アタリ！」、3つとも同じならば「オオアタリ!!」、3つとも違っていれば、「ハズレ」と表示して、持ち点が増えたり減ったりするわけです。

「アタリ！」の場合、かけ点ぶんの点数が持ち点に加えられます。
「オオアタリ!!」の場合は、なんとかけ点の5倍の点を増やしてくれます。ただし、「ハズレ」の場合、かけ点の4倍の点が、持ち点からさしひかれますよ。

```
RUN
        モチテン:  1000

        カケテン:?  100
```

この3つの記号が変化します。

持ち点が0点以下になると、ブザーが鳴り、「ハサンデス!!」と表示してゲームは終りになります。
さあ、あなたはコンピュータ・ルーレットで何点もうけることができますか？　さあ！ルーレットを楽しんでみましょう。

コンピュータ・ルーレットはうまく動きましたか？プログラムが長くなってくるとエラーも多くなってきます。ここで、デバッグの方法をすこし勉強しましょう。コンピュータ・ルーレットがうまく動いた人も、必ず読んでください。

●間違いがどこかわかっている場合

たとえば

　　Syntax error in 100

と出たとします。これは行番号100に、書き間違いがあるという意味ですね。そのときは、

　　LIST　100 RETURN

として、画面にエラーの起こった行を表示し、カーソルコントロールキーを使って、間違った文字をなおしていくのでしたね。実はもっと簡単な方法があるのです。

　　LIST. RETURN

と、**LIST**の後に「.」(ピリオド)を書くのです。こうすると、ベーシックが最後に実行した行番号（つまり、エラーの起こった行番号です）を表示してくれるのです。

```
RUN
        モチテン:  1000

        カケテン:?  200

Syntax error in 100
Ok
LIST.
100 IF L>K TTHEN BEEP:GOTO 60
Ok
▒
```

●間違いがどこにあるのかわからない場合

エラー表示がしっかり画面に出る場合、デバッグは簡単ですね。エラー表示が出ないのに望みの結果が得られないような場合が一番こまります。

たとえば、変数の値がおかしい場合には、プログラムのどこかに **STOP** 文をはさんで **RUN** させます。

STOP 文が実行されると、CTRL + STOP キーが押されたと同じ状態になります。

　　Break in □□□

　　(□□□は、STOP命令の行番号です。)

と画面に表示して、プログラムは実行を停止します。そこで、変数の内容を直接 **PRINT** 文で表示させて調べてみるのです。たとえば、さきほどのルーレット・プログラムの場合に3つの記号がそろっていないのに〝オオアタリ !!〟が出てしまうときに、

　　225 STOP

と書いておきます。**RUN** させれば行番号225で実行は停止しますね。ここで変数Pの値などを**PRINT**して調べてみるのです。もしPの値が負の数だったりしたらこれはあきらかに間違いですね。つまり行番号225より前のプログラムにバグがあるためにこうなったわけです。このように、プログラムの、どこかに **STOP** 文を書いておくことで、どのへんにバグがあるのかけんとうをつけるのです。

STOP 文で実行を停止しても、

CONT RETURN

としてやれば、実行が再開します。

```
RUN
         モチテン:  1000

         カケテン:? 200

               ***

Break in 225
Ok
CONT
```

← とすると実行が再開する。

どこにバグがあるかわかったら、あとは、カーソルコントロールキーで修正すれば、デバッグは終了です。
長いプログラムには、ところどころに**STOP** 文をいれてデバッグしやすくするくふうをしましょう。
もちろんバグがなくなったら、プログラムの中から**STOP** 文は消します。

V 楽しもう グラフィックス &ミュージック

これまでは、画面に表示してきたのは文字が中心でした。今度は、画面に絵を描いたり、コンピュータを楽器と同じように音を出したりする命令を紹介します。
絵を作ってこれを動かすこともできます。
コンピュータを使って絵を描くことをコンピュータグラフィックス、音楽を演奏することをコンピュータミュージックといいます。
さあ、コンピュータを使って、絵と音楽にチャレンジしてみましょう。

COLORで色をつけよう …カラー画面の作りかたとカラーパレットの使いかた

画面にたくさん文字や数字を書いてきましたが、全部白一色で、あまりきれいではありません。画面の色を変える場合には、COLOR命令を使います。

```
COLOR   文字のカラーコード，
        背景のカラーコード，
        境界のカラーコード
```

「，」を忘れずに

0～15の数値、変数、式

・カラーコードは0～15の数値、変数式を使います。

カラーコード

0．透明
1．黒
2．緑
3．明るい緑
4．暗い青
5．明るい青
6．暗い赤
7．水色
8．赤
9．明るい赤
10．暗い黄
11．明るい黄
12．暗い緑
13．紫
14．灰
15．白

●文字の色を変えよう

本機の電源を入れた状態ではカラーコードは15（つまり文字は白）にセットされています。

さっそく色を変えてみましょう。

```
COLOR 1    RETURN
```

と打ち込んでみましょう。画面全体の文字が黒になりますね。これからはキーを押せば、全部黒で画面に書かれます。一度COLOR命令が実行されると、次に別の色でCOLOR命令が実行されるまで、指定された色は変化しません。

もとの白に戻すには

```
COLOR 15    RETURN
```

とすればよいのですね。

●背景色を変えよう

文字の次は背景の色を変えてみましょう。
COLOR命令で背景色を指定することができます。
本機の電源を入れた状態では背景色のカラーコードは4（暗い青）にセットされています。
背景色を緑にするために

```
COLOR , 2  RETURN
```

（↑）文字色を変えない場合には、カラーコードは省略することができますが「,」は忘れないこと

としてみましょう。
背景がきれいな緑になったでしょう。

●境界色を変えよう

COLOR命令で境界色（画面の文字などを表示できる範囲の外側の色）を指定することができます。

```
COLOR , , 2  RETURN
```

（↑↑）「,」を忘れないこと

としてみましょう。画面全体が背景色と同じ緑色になってしまいました。
これではどこからどこまで文字が書けるのかわかりませんね。
もとに戻すことにしましょう。

●もとの状態に戻すには

```
COLOR 15 , 4 , 7  RETURN
```

15：文字色は白
4：背景色は暗い青
7：境界色は水色

とします。ファンクションキーの F 6 を押すと、一回で入力することができますね。

●カラーパレットの使い方

次にCOLOR命令の少し高度な使い方をしてみましょう。

カラーコードで決められた16色以外にも実は、全部で512色もの色をCOLOR命令で出すことができるのです。画面には512色のうち16色を同時に表示することができます。

このときに使うのが、カラーパレットというはたらきです。

画面に表示されている色は全て赤、緑、青の３色からできています。絵の具と同じように、この３つの色をそれぞれ量を変えてまぜあわせて512色を作るわけです。

できあがった色は、絵を描くときと同じようにパレットにとっておきます。コンピュータでは、作った色にパレット番号をつけて記憶しておき、必要なときに取り出して使用します。

好きな色を作って、パレットにとるには次のようにします。

COLOR＝
(パレット番号 ,赤色の量 ,緑色の量 ,青色の量)

0～15の数値 　　それぞれ使う色の量を０～７の数値で

たとえば

赤の量は７ (最大)

緑の量は７ (〃)

COLOR＝ (5 , 7 , 7 , 5)

青の量は５

パレット番号

とすると、パレット５に肌色に近い色ができあがります。

確認してみましょう。

COLOR命令を使います。

COLOR , 5 RETURN

とすると、今までの背景色がパレット５で作った色になりました。今まで、COLOR，５とすると明るい青になったのでしたね。

ところが、パレットを使うと、カラーコードはパレット番号に変わってしまうのです。

512色を表示させるプログラムを作ってみましょう。次のプログラムは、使われている赤、緑、青の量を表示しながら、背景色をいろいろな色に変えるものです。好きな色を作るときに便利なプログラムですからおぼえておいてください。

```
10  CLS
20  FOR I=0 TO 7
30  FOR J=0 TO 7
40  FOR K=0 TO 7
50  COLOR=(1 , I , J , K)
60  COLOR , 1
70  LOCATE 11 , 9 : PRINT "あか
    みどり　あお"
80  LOCATE 6 , 10 : PRINT "COLOR
    =" ; I ; J ; K
90  FOR R=0 TO 500 : NEXT
100 NEXT K , J , I
110 END
```

行番号90のFOR～NEXT命令は、表示している時間を少しあけるためのものです。

500の値を大きくしたり、小さくしたりためしてみてください。

プログラムの中で、よく使われる方法ですからおぼえておいてください。

●もとのカラーコードに戻すには

カラーパレットに色をとっておくと、便利なことは分かりましたね。ところが、元のカラーコードの色を使いたいときには、困りますね。
そんなときのために、COLOR命令には元のカラーコードに戻す方法も用意してあります。

```
COLOR=NEW RETURN
```

または、ただ

```
COLOR RETURN
```

とします。
COLOR，5として確認してみてください。

<起こりやすいエラー>
Illegal function call
(イリーガル ファンクション コール)

COLOR命令の後に書くカラーコードやパレット番号、赤、緑、青の量の数値が定められた数値を超えた場合に起こります。

お絵かきをする前に (SCREEN)

いよいよ次の章から図形を描くためのいろいろな命令を説明していくわけですが、お絵かきをする前にどうしてもやっておかなければならない命令があります。それが、SCREEN命令です。

●画面の種類

本機には、いままであなたが見てきた画面の他に実は８種類の画面があるのです。
つまり、本機は９種類の画面モードを持っています。

0：テキストモード１
　　よこ39字×たて24行
1：テキストモード２
　　よこ29字×たて24行
2：ハイリゾリューションモード
　　よこ256ドット×たて192ドット
3：ローリゾリューションモード
　　よこ64ブロック×たて48ブロック
4：ハイリゾリューションモード
　　よこ256ドット×たて192ドット
5：ビットマップモード
　　よこ256ドット×たて212ドット
6：ビットマップモード
　　よこ512ドット×たて212ドット
7：ビットマップモード
　　よこ512ドット×たて212ドット
8：ビットマップモード
　　よこ256ドット×たて212ドット

聞いたことのない言葉がならんでいますね。「テキストモード」というのは、画面に文字を表示するモードです。何文字画面に表示するかで２種類のテキストモードがあるわけです。今まで使ってきた画面はテキストモード２（よこ29字×たて24行）なのです。
テキストモードはWIDTH命令を使って横方向に表示できる文字数をモード１では最大80字、モード２では32字まで変えることができます。

画面に図などを描くときには、テキストモード以外の画面を使います。ハイリゾリューションモードから下の画面はすべて図形を表示させるための画面モードです。
画面をブロックという単位で分けてあらい絵をかくローリゾリューションモード、細かい点や線がかけるハイリゾリューションモード、さらに点ごとに色などを細かく指定することのできるビットマップモードがあります。各モードの詳しい違いはベーシック文法編をご覧ください。
画面モードのうち、７：ビットマップモードと８：ビットマップモードは、コンピュータのVRAMの容量が128Kバイト以上のものでないと使用できません。お手持ちのコンピュータの取扱説明書の仕様の欄でお確かめください。

●SCREEN（スクリーン）

前ページのように本機には、たくさんの画面が用意されていますが、一度に見ることができるのは、一つの画面だけです。

そこで、これらの画面の切り換えをするのがSCREEN命令なのです。

SCREEN命令は図形を描くプログラムの先頭に必ず書いてください。そうしないと、次の章で説明するPSET、LINE、PAINTなどが、すべてエラーになってしまいます。

SCREEN 画面モード

前ページの各画面モードの先頭に書いてある0～8までの数字

としてやればよいのです。ただしSCREEN命令はプログラムの中で使わないと意味を持ちません。なぜなら画面モードは、コマンド待ちの状態(“OK”とカーソルが表示されていて、ベーシックの命令を受けられる状態）では必ずテキストモードになってしまうからです。

SCREEN命令はこの他に、後で説明するスプライトの大きさや、キーを押したときに出るクリック音、カセットテープのボーレイト（カセットテープレコーダとコンピュータとの間で1秒間に何個のデータをやりとりするかという単位)、専用プリンタの有無などを指定するときに使用します。
詳しくは、ベーシック文法編180ページをご覧ください。

これから、図形を描く命令を説明していきますが、画面モードは、すべてSCREEN 2のハイリゾリューションモードで説明します。

＜起こりやすいエラー＞

●Illegal function call
（イリーガル ファンクション コール）

グラフィック命令をテキストモードで実行しようとすると起こります。

点で絵を描こう

(PSET, STEP, PRESET)

本機は画面に文字や記号の他に難しい図形や絵を描くこともできます。
まずは一番簡単なPSET，PRESET命令から。

横（X軸）方向→
縦（Y軸）方向↓

(0，0)　(255，0)
(128，96)
(0，191)　(255，191)

SCREEN　2の画面です。

●PSET（ピーセット）

画面に絵や図形を描くためには、細かい点が画面に打てなければなりません。
その細かい点をいろいろな形につないでゆくことによって、複雑な絵も描けるわけです。
SCREEN　2の画面には横方向256個、縦方向192個の点を打つことができます。
点の位置は、横（X軸）方向と縦（Y軸）方向の位置を

（横方向の位置，縦方向の位置）

画面の点の位置を数値、変数または式で書きます。

というように表します。
これから、説明する図形を描く命令に共通した位置の書き方ですから、おぼえておいてください。

SCREEN　2の場合（0，0）といったら画面の一番左上の点、(128,96)は画面の真中の点ということになりますね。さて、ではこの位置に点を打つにはどうしたらよいでしょうか?そうです、この時にPSET命令を使うのです。

PSET（横方向の位置，縦方向の位置），カラーコード

カラーコードにはカラーパレットも使えます。
ためしに

```
10  SCREEN 2
20  PSET (128 , 96) , 1
30  GOTO 30
40  END
```

としてみましょう。
画面の中央に黒い点が描かれましたね。
[CTRL]+[STOP]キーでテキストモードに戻ります。
今度はFOR～NEXT命令を使って、点をつないで直線を描いてみましょう。

```
10  SCREEN 2
20  FOR N=0 TO 255
30  PSET (N , 96) , 1
40  NEXT N
50  GOTO 50
60  END
```

とプログラムしてRUNしてみましょう。

画面中央に水平線が描けました。

黒の直線です

●STEP（ステップ）

画面の最後に打った点から、横方向の変化幅と縦方向の変化幅を、画面の位置のかわりに指定して新しく点を打つ位置を指定できる便利な命令がSTEPです。

PSET STEP（横方向の変化幅，縦方向の変化幅）， カラーコード

カラーコードにはカラーパレットも使えます。
STEP命令はPSET命令だけでなく、これから説明するPRESET，CIRCLE，PAINT，LINE命令とも組み合わせて使うことができます。

STEP指定後の点
最後に打った点
縦方向の変化幅
横方向の変化幅

●PRESET（ピーリセット）

PRESET命令は点を消す命令です。

PRESET（横方向の位置，縦方向の位置）， カラーコード

カラーコードにはカラーパレットも使えます。
さきほど作ったプログラムに

```
42  FOR J=0 TO 255
44  PRESET (J , 96)
46  NEXT J
```

を追加して、RUNしてみましょう。
一度描いた黒い水平線が次に消されていくのがわかりますね。
このように背景色と同じ色で点を消していきます。
もし、PRESETに背景色以外の色を指定すると、PSETと同じ働きをします。

<起こりやすいエラー>

●画面上に何もなくなってしまう

PSET命令やPRESET命令の（　）の中に、大きすぎる数や負の数を入れた場合に起こります。
たとえば、

PSET（−100，1500）

のような場合です。
CTRL キーを押しながら STOP キーを押すと、もとの画面に戻ります。

注）テレビ信号の特性により指定した色と異なる場合があります。

直線や長方形、円も簡単に
(LINE, CIRCLE, PAINT)

どんな複雑な図形や絵も、画面上では、結局、点の集まりですから、PSET命令さえあれば、とりあえず何でも画面に描くことができます。
しかし、直線一本を描くにもFOR〜NEXT命令を使わなければならず、大変です。まして画面上にある部分を塗りつぶす場合など非常に時間もかかることになります。
そこで、PSET命令の機能を拡張したいくつかの命令が用意されています。

●LINE（ライン）

LINE命令は、画面のある点から点までを直線で結ぶ命令です。
たとえば

```
10  SCREEN 2
20  LINE (0 , 96)−(255 , 96) , 1
30  GOTO 30
40  END
```

としてみましょう。
PSET命令を使った93ページのプログラムと同じ結果になります。

(0, 96)　(255, 96)

黒の直線です

今度は、

```
25 LINE(0 , 96)−(255 , 96) , 4
```

を追加してください。今描いた直線の上から背景色と同じ色の直線を描いただけです。図形を消す時は背景色と同じ色を図形の上に塗ればよいわけです。

LINE命令は

マイナス記号

```
LINE（始めの位置）ー（終りの位置），
      カラーコード
```

と書くのです。

（始めの位置）はPSET命令で説明したように（横の位置，縦の位置）と、２つの数値を「,」で区切ってならべます。

（終りの位置）についても同じです。

カラーコードにはカラーパレットを使うこともできます。

また、カラーコードを省略して

```
LINE（始めの位置）ー（終りの位置）
```

としてもかまいません。

この場合、COLOR命令で指定されている文字のカラーコードがLINE命令のカラーコードになります。

●LINE命令で長方形を描こう

LINE命令には、もう一つ別な機能があります。

それは画面に長方形を描かせる機能です。

ためしに、次のように打ち込んでRUNさせてみましょう。

```
10  SCREEN 2
20  LINE (50,10)-(200,100),
    8,B
30  GOTO 30
40  END
```

(50, 10)

(200, 100)

どうです？画面に赤い長方形が描かれましたね。

こんどはちょっと変えて

```
25  LINE(50,10)-(200,100),4,B
```

としてRUNさせてみましょう。

今描いた長方形がすぐに消されましたね。

これも背景色と同じ色を長方形の線に塗っただけです。

画面に長方形を描く時は

```
LINE（始めの位置）ー（終りの位置），
      カラーコード，B
```

とすればよいのです。

線を引くだけのLINE命令とちがうのは、一番最後の「B」だけです。この「B」をLINE命令の最後につけることで、画面に長方形を描く命令になるのです。
また、LINE命令で長方形の中を塗りつぶすこともできます。
行番号25を消して

```
20  LINE(50,10)-(200,100),8,BF
```

としてみましょう。
長方形を描くのはさっきと同じですが、長方形の中も赤で塗りつぶしていますね。
次に

```
25  LINE(50,10)-(200,100),4,BF
```

とすれば、赤で塗られた長方形が消えるのはもうわかりますね。

LINE（始めの位置）－（終りの位置）， カラーコード，BF

このように、一番最後に「B」のかわりに「BF」とすれば、長方形を塗りつぶすわけです。

●CIRCLE（サークル）

点の打ちかた、線の引きかた、長方形の描きかたはもうおわかりですね。
今度はCIRCLE命令を使って、円やだ円を描いてみましょう。

CIRCLE(中心の位置),半径,カラーコード

これで円が描けるはずです。
(中心の位置)にはPSET命令で説明したように（横の位置，縦の位置）と2つの数値を「,」で区切って並べます。半径は点の数です。
さっそくやってみましょう。

```
10  SCREEN 2
20  CIRCLE (128,96) , 90 , 15
30  GOTO 30
40  END
```

と打ち込んでください。
画面には大きな白い円が描けますね。

(128, 6)
(38,96)　(128,96)　(218,96)
半径
(点の数が90)
(128,186)

CIRCLE命令の機能はこれだけではありません。だ円、弧、円に関するものなら何だって描けるのです。

CIRCLE（中心の位置），半径，カラーコード，開始角度，終了角度，比率

・比率を書くとだ円を描きます。
　1を中心に、それよりも小さければ平たい円、大きければ縦長のだ円。
　省略すると1とみなし円を描きます。
・開始角度は円弧を描き始める位置を、終了角度は描き終わる角度を0～6.28とラジアンで指定します。

1.57（$\pi/2$）

3.14（π）

0または6.28（2π）

4.71（$3\pi/2$）

開始角度を省略すると0、終了角度を省略すると6.28とみなされます。

ただ読んでいただけではわかりづらいですね。実際にいろいろやってみましょう。

さきほどのプログラムの行番号20を

```
20  CIRCLE (128,96),90,15,,,0.7
```

としてRUNさせてみましょう。
白いだ円が描けます。途中の機能を省略する場合でも「,」だけは書いてください。

(38, 96)　(128, 96)　(218, 96)

次に行番号20を

```
20  CIRCLE(128,96),90,15,1.57,
    6.28,0.7
```

としてRUNしてみましょう。
右上が¼欠けた白い横長のだ円が描かれますね。

注）SCREEN 6で円を描いた場合は横512ドット×縦212ドットのビットマップになっているため、標準は比率を0.5にとってあります。

●PAINT（ペイント）

PAINT命令は、線で囲まれた内側や外側を好きな色で塗りつぶす命令です。

長方形の内側を塗りつぶす場合はLINE命令で簡単にできるのですが、CIRCLE命令で作った円をはじめ複雑な図形を塗りつぶす場合は困りますね。

そこでこの命令があるのです。

次のプログラムをRUNさせてみましょう。

```
10  SCREEN 2
20  CIRCLE (128 , 96) , 90 , 8
30  LINE (128 , 48)-(176 , 128) , 8
40  LINE-(88 , 128) , 8
50  LINE-(128 , 48) , 8
60  PAINT (128 , 32) , 8
70  GOTO 70
80  END
```

ここで行番号40，50はLINE命令の「(始めの位置)」の部分を省略していますが、この場合、前に実行したLINE命令の「(終りの位置)」がそのまま「(始めの位置)」になります。

赤の円と三角形を描き、円と三角形の間を赤く塗りつぶします。

PAINT(128 , 32)，8 は、(128 , 32)の位置から赤(8)で塗りつぶしなさいの意味です。

PAINT（塗り始めの位置），塗る色，境界線の色

「塗り始めの位置から、その境界線で囲まれている部分を塗りつぶしなさい」というのがPAINT命令の意味です。

「境界線の色」は省略することができます。そうすると、「とにかく線で囲まれた部分を塗りつぶしなさい」という命令になります。

ここで使用しているハイリゾリューションモード（SCREEN 2）や（SCREEN 4）では塗る色に境界線の色を指定しなければなりません。

他のグラフィックモード（SCREEN 3 及び SCREEN 5～8）では塗る色と境界線の色を別々に指定できます。

さっきのプログラムの行番号10と60を

```
10  SCREEN 5
60  PAINT (128 , 32) , 15 , 8
```

としてRUNしてみましょう。こんどは赤い線と三角形の間を白く塗りつぶしますね。

PAINT（128 , 32），15 , 8は（128 , 32）の位置から赤い線で囲まれている部分を白く塗りなさいの意味です。

なお、これをSCREEN 2や4で行うことはできません。

ためしに行番号10を

```
10  SCREEN 2
```

としてRUNしてみましょう。

画面全体が白く塗りつぶされましたね。

また、囲まれているはずの線が途中で切れているとそこから色がもれてしまいます。

PAINT命令を使う場合は、必ず線で囲まれていることを確認してください。

<起こりやすいエラー>

●Illegal function call
（イリーガル ファンクション コール）

PAINT命令に使う数値が、規定外の範囲の場合に起こります。

```
PAINT (100 , 100) , 100
```

などが、この種のエラーを起こします。この場合はカラーコードが100で大きすぎます。

スプライトで図形を動かそう

(SPRITE$,PUT SPRITE)

PSET, LINE, PAINT, CIRCLEと図形を描くための色々な命令を学んできました。でもこうした命令で作った図形を動かすのはなかなかめんどうなことなのです。例えば次のプログラムで円を動かしてみましょう。

```
10  SCREEN 2
20  FOR I=1 TO 256 STEP 16
30  CIRCLE (I, 96), 40, 15
40  CIRCLE (I, 96), 40, 4
50  NEXT I
60  END
```

一度描いた円を消して、次の円を描くという動作をして円を動いているように見せているのですが、動きがぎこちないうえに速度も余り速くありません。
たった1個を動かしても不便を感じるのですから、複数の図形を動かすとなるとどれほど大変か想像がつきますね。
しかも動きの速いゲームなどとても作れそうもありません。
こんなときには、スプライトが便利なのです。好みの図形を簡単に作れて、しかもとっても速く画面を動かすことができるのです。

●スプライト

スプライトを学ぶ前に、画面についてちょっと説明しておきましょう。
本機の画面はスプライト面、背景面、境界面と大きく3つに分けることができます。境界面は画面表示には使われないので、ここではスプライト面と背景面について説明しましょう。
まずは背景面から。今まで説明しませんでしたが、PRINT命令による文字の表示や、PSET、LINE、PAINT、CIRCLE命令などの図形表示は、すべてこの画面上で行われてきたのです。　これらの図形を表示させる命令は、さきほど説明したように、図形を動かす目的にはほとんど使えません。つまり、動かない図形に使う画面なので、背景面と呼ぶわけです。
それでは、スプライト面はどんな図形に使われるか、もうおわかりですね。そうです。図形を動かす目的で使うのが、このスプライト面なのです。

スプライト面
(スプライトを描く)
31
30
境界面(表示には使われない)
背景面
(文字やLINE命令で図形を描く)
2
1
0
スプライト面番号
スプライト(1枚のスプライト面に1つだけ描けます)

もう少し、スプライト面について説明しましょう。スプライト面には0から31までのスプライト面番号が表示されるようになっています。この面、1枚1枚に、好きな図形を1個だけ書き込むことができます。ただし、スプライト面が異なれば、同じ図形を32個まで表示できます。スプライト面は、何も描かれていなければ透明なので、テレビの画面には32枚分重なって見えることになります。また、このため、後方にあるスプライト面に描かれた図形と重なると、見えなくなってしまいます。だから奥行のある画面ができるわけです。

さあ、それではスプライトについて説明しましょう。

スプライトというのは、32あるスプライト面に書き込む図形のことなのです。

さきほど説明したように、1枚のスプライト面には1個のスプライトしか書き込めません。言い換えると、1個だけしか表示できないのです。

従って、前ページのプログラムで行ったように一度描いた図形を消して別のところに表示するという手数をかけなくても、1つのスプライト面の中でスプライトを表示する位置を決めてやれば、スプライトはその位置に表示され前の位置のスプライトは消えてしまいます。

つまり、簡単に動かすことができるというわけです。

スプライトの書き方は、LINE命令などで図形を作るときとは、ちょっと違ったやり方をします。次のSPRITE$で説明しましょう。

●SPRITE＄（スプライト ドル）

スプライトを作る命令が、このSPRITE＄です。
スプライトは図形を点の集まりで作ります。まずは、点が縦横それぞれ８×８ドットの大きさのスプライトを作る場合について説明しましょう。まず、８×８ドットのます目を作って、そこに図形を描いてください。空白を０、黒く塗ったところを１として、２進数で表示します。

図形の横方向に１列ずつ２進法でデータを作ります。

１番上の列

０　１　０　０　０　０　１　０

&B01000010と書き表します。
こうして全部で８つのデータができあがりますね。
＆Ｂは、その後に続く数値が２進法ですよということを表す符号です。コンピュータではよく使われる符号です。この他に、16進法もよく使われます。16進法の数値は、＆Ｈという符号の後に０から英文字のＦまでを１ケタとして表します。
私達が使う10進法は通常、符号は付けないことになっています。

全部の図形をデータにしてしまいましょう。

２進数		16進数		10進数
&B01000010	＝	&H42	＝	66
&B10100101	＝	&HA5	＝	165
&B00111100	＝	&H3C	＝	60
&B01011010	＝	&H5A	＝	90
&B11111111	＝	&HFF	＝	255
&B00100100	＝	&H24	＝	36
&B01000010	＝	&H42	＝	66
&B10000001	＝	&H81	＝	129

こうしてできた、８つのデータをCHR＄（　）の中に入れてつなぎ合わせて、スプライト図形を表します。
つまり、図のような図形をスプライトに設定した場合、

```
CHR$(66)+CHR$ (165)+CHR$ (60)+
CHR$ (90)+CHR$ (255)+CHR$ (36)+
CHR$ (66)+CHR$ (129)
```

という文字列で表現するわけです。もちろん、10進数で書かずに、そのまま、２進数や16進数を使って

```
CHR$(&B01000010)+CHR$(&B10100101)+…
CHR$(&B01000010)+ CHR$(&B10000001)
CHR$(&H42)+CHR$(&HA5)+… CHR$ (&
H42)+CHR$ (&H81)
```

としてもよいわけです。

スプライトが8×8ドットでできている場合、本機は256個の異なったスプライトを持つことができます。それぞれスプライト番号で区別します。これをスプライトのパターン番号といいます。ですから、スプライトを作る場合は、図形を表す文字列を作ったら今度はパターン番号も決めてやらなければなりません。
SPRITE$は「コンピュータに図形を表す文字列がスプライトですよ」と宣言する命令です。次のような使い方をします。

SPRITE$(パターン番号)＝図形を表す文字列

それでは、さきほど作った図形（文字列で表されている）をパターン番号0のスプライトとしてみましょう。

```
SPRITE $(0)＝CHR$(66)＋CHR$(165)＋
             CHR$(60)＋CHR$(90)＋
             CHR$(255)＋CHR$(36)＋
             CHR$(66)＋CHR$(129)
```

いよいよ、スプライトが完成です。さっそく作ったスプライトを画面に表示しましょう。それには、PUT SPRITE命令を使いますが説明は後にして、まず次のプログラムを打ち込んでRUNさせてください。

```
10  SCREEN   2 , 0
20  SPRITE$(0)＝CHR$ (66)＋CHR
    $ (165)＋CHR$ (60)＋CHR$ (90)
    ＋ CHR$ (255)＋CHR$ (36)＋CHR
    $ (66)＋CHR$(129)
30  PUT SPRITE 0,(128,96),8,0
40  GOTO 40
50  END
```

さっき描いた図形が赤で表示されましたね。

(128,96)
スプライトの左上が指定した位置になります。

図形が小さくてものたりませんか？。そんな時は、今のプログラムの行番号10を次のように変えてください。

```
10  SCREEN 2 , 1
```

さあ、RUNしてみましょう。さっきのちょうど2倍の大きさの図形を描きましたね。これは8×8ドットのスプライトの1点1点を2倍に拡大しているのです。大きくはなりますが、少し荒い図形になりましたね。

はーい 0番です！
あなたの パターン番号は？

●PUT SPRITE(プット スプライト)

作ったスプライトを画面に表示する命令です。前ページのプログラムの行番号30をよく見てみましょう。

```
30  PUT SPRITE   0 ,(128 , 96) , 8 , 0
```

スプライト面番号　画面位置　カラーコード　パターン番号

この命令は「スプライト面番号0の画面の、画面位置(128, 96)に、カラーコード8(赤)で、パターン番号0のスプライトを描きなさい」という意味なのです。

> PUT SPRITEスプライト面番号, 画面位置, カラーコード, パターン番号

スプライト面番号はもう説明しましたね。0～31の値がとれます。画面位置(128 , 96)という書き方ももうだいじょうぶですね。最初に横方向の位置、次に縦方向の位置を書きます。パターン番号のところには**SPRITE$**命令で作ったスプライトのパターン番号を書きます。**PUT SPRITE**命令を実行する前に、**SPRITE$**でスプライトをつくっておかないと、画面には何も書きませんのでご注意してください。

今度は前ページのスプライトを、動かしてみましょう。

```
10  SCREEN 2 , 0
20  SPRITE$ (0)=CHR$ (66)+
    CHR$ (165)+CHR$ (60)+
    CHR$ (90)+CHR$ (255)+
    CHR$ (36)+CHR$ (66)+
    CHR$(129)
30  PUT SPRITE 0,(0,80),1,0
32  C=INT(RND (1)*15)
34  FOR I=1 TO 256
36  PUT SPRITE 0 , STEP (1, 0),
    C,0
38  NEXT I
40  GOTO 32
50  END
```

書き換えたところや、新しくつけ加えた行には下線が引いてあります。カーソルキーをうまくつかって修正してください。できたら、さっそく**RUN**してみましょう。

```
>>>>>      >>>>>>>
```

さきほど作ったインベーダーのような図形が画面の左から右へ動いていますね。

STOP キーを押すと、止まったりまた動き出したりします。

PUT SPRITE命令で、画面位置を指示するところに

> STEP (横方向の変化幅, 縦方向の変化幅)

と置くと、「指定したスプライト面の以前の位置から、横方向の変化幅、縦方向の変化幅だけスプライトを動かしなさい」という意味になるのです。スプライトを描くだけなら(横方向の位置、縦方向の位置)と書けばよいのですが、スプライトを動かしたい場合は、**STEP**を使います。

●もっと大きなスプライトを描いてみよう

8×8ドットのスプライトを拡大して表示する方法は説明しましたね。今度は、最初から16×16ドットのスプライトを作る方法を説明しましょう。

16×16ドットのます目を用意して、そこに図形を描いてください。空白を0、黒く塗ったところを1として2進数になおすまでは、前と同じですね。

```
0000 0000 | 0000 0000
0000 0000 | 0000 0000
0000 0001 | 1000 0000
0000 0011 |
0000 1111 |
0011 1101 |
1110 1010 |
1110 1010 |
----------+----------
1111 1111 |
```

できた2進数をキャラクタコードになおして、文字列にしていく点も前と同じです。ただ、ちょっと気をつけなければならないのは、文字列にしていく順序です。

図のように、全体を8ドットずつの4つに分けます。

8ドット

①	③
②	④

8ドット

左上の8ドットのひとかたまりから、さきほどの8ドットの場合と同じように、横方向に1列ずつデータを作っていきます。

左上が終わったら、左下、右上、右下の順にデータを作っていきます。

全部で32個のデータができあがるわけですね。

作った2進数はこの順序で文字列にしていくんだよ

①	⑰
②	⑱
③	⑲
④	⋮
⑤	⋮
⑥	⋮
⑦	㉓
⑧	㉔
⑨	㉕
⑩	⋮
⋮	⋮
⑯	㉜

さあ、それでは、さっそく作って動かしてみましょう。次のプログラムを打ち込んでください。
データの量が多くなりますので、行番号30と120から150までにREAD, DATA命令という特殊な命令を使ってあります。この命令の詳しい使い方は後で説明しますので、とりあえずこのとおり打ち込んでください。

```
10  SCREEN 2 , 3
20  FOR I=1 TO 32
30  READ A:B$=B$+CHR$(A)
40  NEXT I
50  SPRITE$(0)=B$
60  PUT SPRITE 0 , (110 , 0) , 1 , 0
70  C=INT(RND(1)*15)
80  FOR I=1 TO 70
90  PUT SPRITE 0 , STEP( 0 , 1 ), C, 0
100 NEXT I
110 GOTO 70
120 DATA &B00000000 , &B00000000 , &
    B00000001 , &B00000011 , &B00001111 , &
    B00111101 , &B11101010 , &B11101010
130 DATA &B11111111 , &B01111111 , &
    B00011001 , &B00001111 , &B00000111, &
    B00000000 , &B00000000 , &B00000000
140 DATA &B00000000 , &B00000000 , &
    B10000000 , &B11000000 , &B11110000 , &
    B10111100 , &B01010111 , &B01010111
150 DATA &B11111111 , &B111111110 , &
    B10011000 , &B11110000 , &B11100000 , &
    B00000000 , &B00000000 , &B00000000
160 END
```

120のDATAが左上1/4、130のDATAが左下1/4、140のDATAが右上1/4、150のDATAが右下1/4を示しています。

行番号10でSCREEN 2, 3としている点に注意してください。スプライトの大きさを16×16に設定していますね。こうしないと、8×8ドットのスプライトを描いてしまいます。16×16ドットのスプライトは全部で64個までつくれます。また、行番号120からのDATA文は少々こみいっています。これはデータを2進法で書いたためですが、これらを10進法や16進法になおしてプログラムしてもかまいません。少々手間がかかりますが…
さあ、RUNしてみましょう。

大きなUFOが色を変えながら画面の上から下へ動いていきますね。
スプライトはゲームだけでなく、お絵描きソフトなどのカーソルの替わりとして使われたり、アニメーションなどにも使うことのできる便利な機能です。

本機には、ここで説明したスプライトの他に、カラースプライトという便利な命令も内蔵されています。
ただし、画面表示やコンピュータについて少し詳しい知識を必要としますので、基礎編では説明をしません。
詳しくお知りになりたい方は、文法編のCOLOR SPRITE、COLOR SPRITE$ 命令をご覧ください。

音だって出せるんだ

(BEEP, PLAY)

本機は画面に絵や文字を書くだけではありません。音楽を演奏させることもできるのです。まずは、音を出すだけのBEEPから。

●BEEP(ビープ)

BEEPは、ブザーを鳴らす命令です。

```
BEEP [RETURN]
```

としてみてください。「ピーッ」とブザーが鳴りましたね。今度は

```
10 FOR N=1 TO 10
20 BEEP
30 NEXT N
40 END
```

とプログラムを作ってRUNさせてみましょう。「ピッ，ピッ……」と10回ブザーが鳴りますね。

このBEEPは、音の高さも音の長さも変えることができないので、音楽演奏には向いていません。本機で音楽を演奏させたい人は、次のPLAY命令をお使いください。

●PLAY(プレイ)

```
PLAY "CDEFRGAB" [RETURN]
```

と打ち込んでみましょう。まず、ドレミファと音を出し、少し休んだあとソラシと演奏しますね。この「A」～「G」の文字は、それぞれ

文字	C	D	E	F	G	A	B
音階	ド	レ	ミ	ファ	ソ	ラ	シ

というふうに音階を表しているのです。また「R」は休符を意味します。
このように、「A」～「G」までの文字や「R」を「"」でかこんでPLAYの後に書くと、曲が演奏できるのです。

```
PLAY "曲を表わす文字列"
```

今度は、音の長さを変えてみましょう。それには「A」～「G」、「R」のすぐ後に音の長さを表わす数字を書けばよいのです。

```
PLAY "C1D2E4F8R1G16A32B64" RETURN
```

ドレミファと、だんだん音を短かくしていき、しばらく休んだのちソラシと、もっと短い音で演奏していくのがわかりますね。**音の長さは1～64まで細かく指定することができます。**ただし、4が4分音符、8が8分音符というように、実際の音符に対応しているのは1、2、4、8、16、32、64だけです。数が大きくなると音が短くなることに注意してください。

さて、それでは符点4分音符を出すときにはどうすればよいでしょうか？

```
PLAY "C4." RETURN
```

というふうに、数字の後に「.」(ピリオド) をつけてしまうのです。

ここまでで、音楽演奏のための基本的なことはほぼ説明しました。**PLAY**命令はもっと簡単に、そしてもっといろいろな音を出すために、このほかにもいろいろな機能を持っているのです。

イ) #、+、－ ……シャープ、フラット

曲を演奏するためには半音上げる（シャープ）ことや、半音下げる（フラット）ことも必要ですね。

#、+：「A」～「G」のすぐ後につけると半音上げた音を出す。
－：「A」～「G」のすぐ後につけると半音下げた音を出す。

「#」、「+」、「－」はそれぞれピアノの黒い鍵盤に当たります。ただし、例外としてC－はB、E＋はF、F－はE、B＋はCとなって現在指定しているオクターブの音となります。

たとえば4分音符ドのシャープの音を出すときは

```
PLAY "C+4" RETURN
```

とすればよいのです。「#」、「+」、「－」は「A」～「G」と音の長さの間に入れるということを覚えておいてください。

ロ) L…音の長さの指定

今までは、「A」～「G」の後に数字を書いて、音の長さを指定しましたね。でもそれではちょっとめんどうなこともあります。全部同じ音の長さで曲を演奏したいときなどがそうです。
たとえば

```
PLAY "C4D4E4F4G4A4B4" RETURN
```

としたい場合などです。全部に4分音符を指定するのはめんどうですね。
「A」～「G」、「R」の後に数字を書かないでおくと、ある決まった長さの音を出すことができます。その長さを指定するときに「L」という文字を使います。さっきの例を使うと

```
PLAY "L4CDEFGAB" RETURN
```

とすればよいのです。
つまり、音の長さには「音階を表わす文字の次に数字があれば、その長さで音を出す。なければ、「L」の後に書いた数字の長さで音を出す。」という規則があるのです。この「L」は一度指定すると、次の「L」がくるまで、ずっとそのままになっています。たとえば「L8」と指定したら、電源を切るまで、「A」～「G」、「R」の後に数字のないものは8分音符で音を出すというわけです。また、電源を入れたときは「L4」を指定したと同じ状態になっています。

ハ）O…オクターブの指定

これまでの説明では、１オクターブぶんの音しか出すことができません。もっと高い音や低い音を出すにはどうしたらよいでしょうか？
それには「O」と数字を音階を表わす文字の前に入れます。

```
PLAY  "O3CDEFGABO4CD
 EFGABO5CDEFGABO4"
                    RETURN
```

としてみましょう。３オクターブ分ドレミファソラシと演奏されたでしょう。「O」のすぐ後に書いた数字でオクターブを指定することができるのです。数字は１～８、つまり８オクターブの音を本機は出せるのです。「O」も一度指定すると、次の「O」までその値を保ちます。また、電源を入れたときは「O４」のオクターブになっています。
ここまで説明した命令を使って、曲を１つ作ってみましょう。

```
10  FOR N=0  TO  3
20  PLAY  "L4C.L8DL
           4E.L8CL4
           ECL2E"
30  NEXT  N
40  END
```

ここで、C.E.は、それぞれC4.E4.と同じです。

ニ）T…テンポの指定

曲全体を速く演奏したり、遅く演奏したりするときには「T」という文字を使います。さっき作ったプログラムの20行目を次のように書き換えましょう。
T240を追加するだけです。

```
20  PLAY  "T240L4C. L8D L4E.
           L8C L4EC L2E"
```

これでRUNさせてみましょう。さっきの２倍の速さで演奏していますね。
「T」のすぐ後の数字で、曲全体の速さを変えられるわけです。数字は32～255の範囲で使えます。数字が大きい方が速く演奏します。また、この「T」も次の「T」がくるまで、ずっとそのままになっています。電源を入れたときは「T120」の速さと同じです。

ホ）V…ボリューム

次は音の大きさも変えてみましょう。それには「V」と数字を音階を表わす文字の前に入れます。

```
PLAY  "L1V3CV6DV9EV12
       FV15G"  RETURN
```

としてみましょう。だんだん音を大きくしながらドレミファと演奏していきます。
「V」の後に書ける数字は０～15までです。数が大きくなるほど音も大きくなります。「V０」を指定すると音を出しません。
電源を入れたときは「V８」の大きさです。
ゲームなどでは、V12～V14程度で使うと適正レベルで、プログラムが組めます。

へ）S、M…音色を変える

「S」、「M」と数字を組み合わせて使うと、音色を変えることができます。たとえば次のように打ち込んでください。

```
PLAY "S10M3000CDEFGAB"
```
RETURN

「S」、「M」の後の数字をいろいろ変えてみましょう。様々な音色を出すことができます。Sは0〜15、Mは1〜65535の間の数字で指定することができます。

本機は、2重和音、3重和音も出せます。
より詳しい説明は、ベーシック文法編190ページをご覧ください。

Ⅵ 本格的にプログラミング

ベーシック基礎編の最後に、コンピュータらしい本格的な命令をいくつか紹介しましょう。
漢字を使ったり、データを使う命令です。
ベーシックだけでなく、コンピュータに直接命令を伝えることのできる機械語についても、少し説明をすることにしましょう。
基礎的な命令の説明はここでおしまいです。
後は自分でいろいろなプログラムを作って、ベーシックの腕前を向上させてください。

文字と数字を交換しよう
…VAL関数，STR$関数の使い方

文字列と数はまったく別のものですから組み合わせて計算することはできないと説明しました。

A$＝"123" RETURN

では、A$は123という数ではなく、「123」という文字列なのです。こうした文字列を数値に、また数値を文字列に換えることができます。本機を時計にして使うときなどにきっと便利ですよ。
そんなときに使うのが VAL と STR$ です。

●VAL（バリュー）関数

VAL を使うと数字で書いてある文字列を数値になおすことができます。

VAL（文字列か文字変数）

次のように打ち込んでみましょう。

```
10  A$＝"12345"：B$＝"54321"
20  C$＝A$＋B$
30  PRINT  C$
40  C＝VAL(A$)＋VAL(B$)
50  PRINT  C
60  END
```

さっそく　RUN　してみましょう。図のような画面になりましたね。行番号40の「＋」は、たし算の「＋」であることに注意してください。

```
Ok
RUN
1234554321  ←これがPRINTC$
 66666      ←こっちがPRINTC
Ok
■
```

数値にすると「－」をいれる部分が1文字あくよ。

●STR$（ストリング ドル）関数

このSTR$関数はVAL関数とはまったくの反対で、数値を文字列に換える働きをします。

STR$（数値か数値変数）

式も書けます。　　かっこを忘れずに

例）STR$(23＊45)

次のプログラムを打ち込んで、RUN　させてみましょう。

```
10  A=12345 : B=54321
20  C=A+B
30  C$=STR$(A)+STR$(B)
40  PRINT  C
50  PRINT  C$
60  END
```

行番号30の「+」は、計算するときのたし算の記号ではなくてここでは文字列をつなげる「+」であることに注意してください。

```
Ok
RUN
 66666        ←PRINT Cの結果
 12345 54321  ←PRINT C$の結果
Ok
█
```

もともとが数値なので1文字分あいてしまう

〈起こりやすいエラー〉

● Type mismatch(タイプ ミスマッチ)

51ページ、264ページをよく読んでください。

キャラクターコードのお話

(CHR$関数，ASC関数)

本機では、たくさんの文字をキーボードから入力して表示することができますが、コンピュータには、キーボードにある以外の文字や図形も表示させることができるのです。

文字や図形に番号を割りあてた表がコンピュータの内部にあるのです。この、割りあてられた番号を、キャラクターコードといいます。

キャラクターコード表

2バイトコード（上位1バイト01H）

	4	5
0		π
1	月	
2	火	
3	水	
4	木	
5	金	
6	土	
7	日	
8	年	
9	円	
10	時	
11	分	
12	秒	╳
13	百	大
14	千	中
15	万	小

1バイトコード（縦：下位4ビット、横：上位4ビット）

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		0	@	P	'	p	♠			ー	タ	ミ	た	み
1	!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
2	"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
3	#	3	C	S	c	s	◆	う	」	ウ	テ	モ	て	も
4	$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や
5	%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ
6	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
10	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
11	+	;	K	[	k	{	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
12	,	<	L	¥	l	|	ゃ	し	ャ	シ	フ	ワ	ふ	わ
13	-	=	M	]	m	}	ゅ	す	ュ	ス	ヘ	ン	へ	ん
14	.	>	N	^	n	~	ょ	せ	ョ	セ	ホ	゛	ほ	
15	/	?	O	_	o		っ	そ	ッ	ソ	マ	゜	ま	■

■←カーソル

キャラクターコードは上の表から次のような式で求めます。

(横方向の数字)×16+(縦方向の数字)＝（求めるキャラクターコード)

たとえば

Qは5×16+1＝81となります。

ただし、左の表（2バイトコード）の場合は、次のCHR$（キャラクタードル）関数をご覧ください。

コンピュータはこのキャラクターコードの表から「A」という文字は65番、「！」という記号は33番というようにさがしてきて表示しているのです。

キャラクターコードを扱う命令に、CHR$と、ASCがあります。

●CHR$（キャラクタードル）関数

CHR$は、キャラクターコードで文字を探す関数です。

CHR$（数値か数値変数）

PRINT　CHR$（65）RETURN

と打ってください。「A」を画面に表示しますね。キャラクターコード表の65番目の文字はAであったわけです。

```
Ok
PRINT CHR$(65)
A
Ok
■
```

(65) を他の数値に変えていろいろな文字や図形を表示させてみましょう。

2バイトコード（グラフィック文字）の場合は、

CHR$(1)＋CHR$（数値か数値変数）

例えば、月は

CHR$(1)+CHR$（4×16+1）です。

この CHR$ 関数を使って、キャラクターコード表を画面に書いてみましょう。

```
 10 CLS
 20 FOR X=2 TO 15
 30 FOR Y=0 TO 15
 40 LOCATE X,Y
 50 IF X=7 AND Y=15 THEN
    PRINT CHR$(32):GOTO70
 60 PRINT CHR$(X*16+Y)
 70 LOCATE 0,Y
 80 PRINT CHR$(1)+CHR$(4
    *16+Y)
 90 LOCATE 1,Y
100 PRINT CHR$(1)+CHR$(5
    *16+Y)
110 NEXT Y
120 NEXT X
130 END
```

このプログラムを RUN させてください。16×16で256個の文字や記号を画面に表示します。

注）キャラクターコードの1～31と127は、コントロールコードに使用されています。詳しくは、262ページのコントロールコード一覧表をご覧ください。

●ASC（アスキー）関数

ASC は CHR$ とは逆に、文字からキャラクターコードを探し出す関数です。

ASC（文字列か文字変数）

PRINT　ASC（"A"）RETURN

と打ってください。「A」のキャラクターコード65を画面に書いてくれます。

また、

PRINT　ASC（"ABC"）RETURN

などとかっこの中に文字列を書いた場合、"ABC"の先頭の文字「A」のキャラクターコードを表示します。この場合、65ですね。

ただし、2バイトコードのキャラクターコードの場合には、すべて1を返します。

```
Ok
PRINT ASC("A")
 65
Ok
PRINT ASC("ABC")
 65
Ok
■
```

〈起こりやすいエラー〉

●Illegal function call
（イリーガル ファンクション コール）

0～255範囲外の数を使った場合に起こります。

CHR$（300）

などとした場合がそうです。

デジタル時計を作ろう

(TIME, SET, GET)

本機では、文字を書いたり、絵を描いたりすることができましたね。それだけではありません。
コンピュータを時計として使うこともできるのです。
このとき使うのがTIME，GET，SETです。

●TIME（タイム）

TIMEは、60倍の秒単位で時間を教えてくれるシステム変数です。次のように打ち込んでください。

```
PRINT TIME  RETURN
```

どんな結果がでましたか？　この結果は、コンピュータ電源を入れてからどのくらい時間がたったかを60倍の秒で示しているものです。

```
Ok
PRINT TIME
 652
Ok
█
```

ここは、ほかの表示がでても間違いではありません。

TIMEの値は、あなたが自由に変えることができます。つまり、時計の時刻をセットするようにTIMEを変えることができるのです。
例えば

```
TIME=0       RETURN
PRINT TIME   RETURN
PRINT TIME   RETURN
```

とすると、TIME＝0にして時点からどのくらい時間がたったか教えてくれます。だから、実際の秒は60で割ればよいのです。

```
TIME=0
Ok
PRINT TIME
 3600
Ok
PRINT TIME
 10800
Ok
```

3600：ここまでで60秒たったことを意味する
10800：ここまでで180秒たったことを意味する

ここはほかの表示がでても間違いではありません

今度は、このTIMEを使って、簡単な時計を作ってみましょう。

```
10  CLS
20  TIME=0
30  LOCATE 0,10
40  PRINT SPC(6);"分";
    SPC(6);"秒"
50  S=INT(TIME/60)MOD 60
60  M=INT(TIME/3600)
70  LOCATE 2,10:PRINT M
80  LOCATE 9,10:PRINT S
90  GOTO 50
100 END
```

RUNさせると、そのときからの経過時間を

```
11分18秒
```

ここは、ほかの表示が出ても間違ではありません。

とあなたに教えます。

ただし、このプログラムでは、18分12秒までの表示で、これを過ぎると0分0秒になり、もう一度最初からカウントを始めます。

TIMEは、コンピュータが例えば数値を画面に表示する、プリンタに出力する、計算をするというように何か仕事をするときに、そのタイミングをはかるために専用に持っている時計機能（内蔵クロックといいます）を使って、私達に時間を教えてくれます。

ですから、私達がふだん使っている時計とは目的が違ってあまり大きな数値になる時間まではカウントする必要がないのです。

そこで、本機の持つもうひとつの時計を紹介することにしましょう。

本機を初めてご使用になるときに、本機に電池を入れたことを覚えていますか？

この電池は、本機の持つもうひとつの時計のためのものなのです。こちらの時計は、TIMEと違って、ふだん私達が使っている時計と同じ働きをします。ただし、電気で働いていますから本機の電源を切っても時計が止まらないように電池をいれておくのです。

では、この時計を使ってみることにしましょう。

●GET TIME（ゲットタイム）

時計を見るときに使う命令です。この命令で直接時間を知ることはできません。GET　TIME命令は時間を文字変数に代入します。従って時間を知るためには、その文字変数の内容をPRINT命令などで画面に表示させればよいのです。

```
GET TIME 文字変数
```

文字変数だから＄記号を忘れないでください。

次のプログラムを打ち込んでください。

```
10  GET TIME A$
20  LOCATE 10,12 :PRINT A$
30  GOTO 10
```

さっそくRUNさせてみましょう。

```
10 GET TIME A$
20 LOCATE 10,12:PRINT A$
30 GOTO 10
RUN
```

→ 20：15：16

ここは違った表示が出ても間違いではありません。

正確な時間を示していますか？
もし違っていたら、時計を合わせておかなければなりません。
CTRL + STOP でプログラムを止めてください。

●SET TIME（セット タイム）
時計を合わせるための命令です。

SET TiME”時間：分：秒”

24時間制の時刻を入れてください。
(例 午後5時→17時)

必ず2ケタの数字で
(ゼロの場合は00)

例えば、午後8時5分00秒に合わせるときには、

SET TIME “20：05：00” RETURN

とします。画面に命令を書いておいてから、正確な時計を見ながら、RETURN キーを押すとよいでしょう。

●GET DATE（ゲット デート）
本機の時計には、カレンダーもついています。
現在の日付を見るには、GET DATE命令を使います。GET TIME命令と全く同じ使いかたをします。

GET DATE 文字変数

日付を知るためには、この命令の後、PRINT命令などで文字変数の内容を表示させればよいわけです。

●SET DATE（セット デート）
GET DATE命令で日付が正しくなかった場合にはカレンダーを正しい日付に直しておかなければなりません。そのときに使う命令がSET DATE命令です。

SET DATE”年／月／日”

西暦年の下2ケタ
(1985年→85)

必ず2ケタの数字で
(2月の場合は02というように)

●デジタル時計を作ろう

本機の時計は、私達がふだん使っている時計と同じ働きをするだけでなく、プログラムのなかで使うことができるという便利な特長があります。

少し長くなりますが、デジタル時計プログラムを作ってみることにしましょう。

```
10 CLS
20 DIM NU$(15,15)
30 '
40 PA$(1)="  ●  "
50 PA$(2)="●●●●●"
60 PA$(3)="●    "
70 PA$(4)="    ●"
80 PA$(5)="●   ●"
90 '
100 FOR I=0 TO 9
110 FOR K=1 TO 14 STEP 2
120 READ PN
130 NU$(I,K)=PA$(PN)
140 NU$(I,K+1)=PA$(PN)
150 NEXT K,I
160 '
170 DATA 2,5,5,5,5,5,2
180 DATA 1,1,1,1,1,1,1
190 DATA 2,4,4,2,3,3,2
200 DATA 2,4,4,2,4,4,2
210 DATA 5,5,5,2,4,4,4
220 DATA 3,2,3,2,4,4,2
230 DATA 3,3,3,3,2,5,2
240 DATA 2,5,4,4,4,4,4
250 DATA 2,5,5,2,5,5,2
260 DATA 2,5,2,4,4,4,4
270 GOSUB 510
280 GET TIME TI$
290 GOSUB 360
300 '
310 GET TIME TI$
320 IF RIGHT$(TI$,2)="00" THEN GOSUB 360
330 IF SP=1 THEN SP$=" " ELSE SP$="◆"
340 GOSUB 690
350 GOTO 300
360 '
370 IF RIGHT$(TI$,5)="00:00" THEN GOSUB 590
380 H1=VAL(MID$(TI$,1,1))
390 H2=VAL(MID$(TI$,2,1))
400 M1=VAL(MID$(TI$,4,1))
410 M2=VAL(MID$(TI$,5,1))
420 FOR I=2 TO 8 STEP 6
430 IF I=2 THEN NU=H1 ELSE NU=H2
440 GOSUB 760
450 NEXT I
460 FOR I=16 TO 22 STEP 6
470 IF I=16 THEN NU=M1 ELSE NU=M2
480 GOSUB 760
490 NEXT I
500 RETURN
510 '
520 GET DATE DA$
530 LOCATE4,3:PRINT DA$
540 LOCATE2,3:PRINT "19"
550 LOCATE6,3:PRINT "年"
560 LOCATE9,3:PRINT "月"
570 LOCATE12,3:PRINT "日"
580 RETURN
590 '
600 IF TI$="00:00:00" THEN GOSUB 510
610 GET TIME PL$
620 PL=VAL(LEFT$(PL$,2))
630 IF PL>12 THEN PL=PL-12
640 IF PL=0 THEN PL=12
650 FOR I=1 TO PL
660 PLAY "S0M20000O2A1"
670 NEXT I
680 RETURN
690 '
700 LOCATE14,11:PRINT SP$
710 LOCATE14,12:PRINT SP$
720 LOCATE14,15:PRINT SP$
730 LOCATE14,16:PRINT SP$
740 IF SP=1 THEN SP=0 ELSE SP=1
750 RETURN
760 '
770 FOR K=7 TO 21
780 LOCATEI,K:PRINT NU$(NU,K-7)
790 NEXT K
800 RETURN
```

配列って何？

(DIM)

たとえば26個の変数に値を入力するプログラムを考えてみましょう。

```
10  INPUT  AA
20  INPUT  AB
30  INPUT  AC
        :
        :
        :
260 INPUT  AZ
        :
        :
```

こんなふうに書いていったら大変です。変数を入力する部分だけで、26行ものプログラムになってしまいますね。もっと簡単に書く方法はないものでしょうか？ベーシックにはDIM命令というものがあり、うまく使うとこのようなプログラムを非常に簡単に書くことができます。

```
10  DIM  A(25)
20  FOR  N=0 TO 25
30  INPUT A(N)
40  NEXT N
        :
```

26行あった部分が、わずか4行ですんでしまいました。

●DIM（ディメンジョン）

```
DIM X(25)
```

上の命令は「Xという名前の箱を用意しなさい。ただし、その箱には26の引出しがあって、0～25までの番号がついていますよ。」という意味です。

その引出しの中に数値をしまうことができます。たとえば、１番目の引出しに10という数をしまいたいなら、

X(1)=10

としてください。ほんとうに入っているかどうか確かめてみるには、

PRINT X(1)

とすればよいのです。
いまの説明をプログラムにしてRUNさせると次のようになります。

```
10 DIM X(25)
20 X(1)=10
30 PRINT X(1)
40 END
RUN
 10
Ok
```

なんとなくわかってきましたか？
ここで使ったX( )が配列と呼ばれるもので、引出しつきの変数なのです。

つまりDIMは、変数の引出しの数を宣言する命令で、これを配列の定義といいます。

添字といいます。

DIM 変数名(数値か変数か式)
, 変数名( ), …

配列をたくさん使うときには、並べて書けます。

変数名のつけ方は、27ページの「変数名の３つの規則」をよく読んでください。

配列を使うときは、変数名の後に番号をかっこではさんでつけます。（この番号を添字といいます）つまり

Xという引出しの	0番目	……	X(0)
〃	1 〃	……	X(1)
⋮	⋮	……	⋮
〃	25 〃	……	X(25)

かっこの中の番号は、 DIM 命令で使ったかっこの中の数より大きくなってはいけません。また、プログラムの中では DIM 命令は配列を使うときより先に書かれていなければなりません。

次のプログラムは、20個の数字を入力して、その数の平均値を求めるものです。

```
 10  PRINT  "20コノ カズヲ イレテク
              ダサイ"
 20  DIM   D(20)
 30  S = 0
 40  FOR  J =1  TO  20
 50  PRINT  J;: INPUT
               "バンメ";D(J)
 60  S = S + D(J)
 70  NEXT  J
 80  A = S/20
 90  PRINT  "ヘイキン  ハ";A;"デス"
100  END
```

行番号20で、0～20の21個の引出しをもつ変数Dを使えるようにし、行番号40で、FOR～NEXT命令でくり返しながら、Dに1つずつ数値を入れていきます。RUNさせてみましょう。

```
RUN
20コノ カズヲ イレテクダサイ
 1 バンメ? 10
 2 バンメ? 20
 3 バンメ? 15
      :
      :
 20バンメ ? 101

ヘイキン ハ 32.5 デス
Ok
■
```

●複雑な配列にチャレンジ

今までに説明した配列は引出しが縦1列だけでした。本機は、縦、横に何列もの引出しが並んだ配列を使うこともできます。

たとえば、

```
10 DIM  Y (5, 10)
```

と打ち込んでみましょう。この命令は「Yという名前の箱を用意しなさい。ただしその箱には、縦に6列、横に11列の引出しがあって、それぞれの0～5、0～10までの番号がついていますよ」という意味です。

縦横2つの方向に引出しをもっているので、このような配列を2次元配列と言います。まえで説明した縦方向にしか引出しをもたない配列を1次元配列と言います。

2次元配列を使うときには、

変数名（縦の引き出しの位置，横の引き出しの位置）

として、かっこの中に2つの数値を指定する必要があります。たとえば、

```
10  DIM Y (5, 10)
20  Y(1, 1)=15 : Y(2, 5)=17
30  PRINT  Y(1, 1), Y(2, 5)
40  END
```

のように使うわけです。

```
10 DIM Y(5,10)
20 Y(1,1)=15:Y(2,5)=17
30 PRINT Y(1,1),Y(2,5)
40 END
RUN
 15             17
Ok
```

次のプログラムは、３人の英語、数学、国語のテストの点数を入力するときれいに並べて表示します。

```
10  DIM  TEST (3, 3)
20  PRINT  "えいご, すうがく, こくごの じゅんばんで いれること!"
30  FOR  K=1  TO  3
40  PRINT  "NO";K;:INPUT "の とくてんは"; TEST (1, K), TEST (2, K), TEST (3, K)
50  NEXT  K
60  CLS
70  PRINT SPC(6); "えいご" ; SPC(3); "すうがく" ; SPC(4) ; "こくご"
80  FOR  K=1  TO  3
90  FOR  I=1  TO  3
100 LOCATE  0, 3 * K : PRINT  K
110 LOCATE (I * 8-3), 3 * K : PRINT  TEST (I, K)
120 NEXT  I
130 NEXT  K
140 END
```

RUN させたらひとりずつ、英語、数学、国語の点数を「，」でくぎって入れてください。

```
RUN
えいご゛，すうか゛く，こくご゛の じ゛ゅんは゛んで゛
いれること！
NO 1 の とくてんは? 50,45,58
NO 2 の とくてんは? 85,79,92
NO 3 の とくてんは? 70,60,72

        えいご゛        すうか゛く      こくご゛

1       50             45             58
2       85             79             92
3       70             60             72

Ok
■
```

●文字の配列もあるよ

数値変数と同じように、文字変数にも配列を作ることができます。

```
10 DIM S$(2)
```

としてみましょう。これでS$(0)～S$(2)までの文字の変数が3つとれたわけです。次に

```
20 S$(0)="わたしは ":S$(1)=
   "パソコン":S$(2)="です"
30 FOR N=0 TO 2:PRINT S$
   (N);:NEXT N
40 END
```

としてRUNさせてみましょう。ちゃんと配列に文字が入っていますね。

```
10 DIM S$(2)
20 S$(0)="わたしは ":S$(1)="ハ゜ソコン
 ":S$(2)="て゛す"
30 FOR N=0 TO 2:PRINT S$(N);:
NEXT N
40 END
RUN
わたしは ハ゜ソコン て゛す
Ok
■
```

文字変数でも、配列にすることができます。

〈起こりやすいエラー〉

●Redimensioned array
（リディメンジョンド アレイ）

すでに宣言した配列をもう一度宣言した場合に起こります。たとえば、

```
DIM A(20)
```

という命令を2回実行した場合などです。

●Subscript out of range
（サブスクリプト アウト オブ レンジ）

配列の添字が、宣言した範囲をこえている場合です。たとえば、

```
DIM A(20)
```

と配列を定義したのに

```
A(21)
```

などを使った場合です。

READ～DATAでデータを使おう

(READ, DATA, RESTORE)

変数に値を入れるときには、A＝0やB＝10のような代入文のほかに、INPUT 命令を使ったりしました。実はもう一つ方法があります。それが READ～DATA 命令です。

READ X, Y, Z	……いくつかの変数
DATA 10, 20, 30	……それに対する数値

● READ ～ DATA (リード～データ)

次のプログラムを RUN してみましょう。

```
10  READ  X, Y, Z
20  PRINT  X + Y + Z
30  DATA  10, 20, 30
40  END
```



```
RUN
 60
Ok
```

行番号10の READ　X, Y, Z は「 X, Y, Z の値を読み込みなさい」という意味です。読み込まれる X, Y, Z の値はどれかというと行番号30の DATA 命令後の10と20と30です。

このようにして READ～DATA 命令を使って変数に数値や文字を入れることができます。決まっている値を変数に入れる場合にとても便利な命令です。 INPUT 命令は、RUN させるたびに変数の値を入力しなければならないので変数の値が決まっている場合は、かえって不便なのです。

READ～DATA 命令と、前に学んだ LINE を使って、ちょっとした図形を描いてみましょう。

```
 10  SCREEN  2
 20  READ  X0 , Y0, X1, Y1
 30  LINE(X0, Y0)-(X1, Y1), 8,
     BF
 40  FOR  I =9  TO  14
 50  READ  X, Y
 60  LINE  -(X, Y), I, BF
 70  NEXT  I
 80  GOTO  80
 90  END
100  DATA  100, 100, 125, 120
110  DATA  150, 80, 75, 60,
           55, 140, 180, 160,
           200, 40, 55, 20
```

灰(14)
暗い黄(10)
明るい黄(11)
明るい赤(9)
赤(8)
紫(13)
暗い緑(12)

このプログラムでは、DATA より前の行に END がありますが、DATA は、プログラムのどの部分に書いてもよいのです。

また、READ～DATA 命令で、文字列を文字変数に読み込ますこともできます。次のプログラムを打ち込んで、RUN させてみましょう。

```
 10  CLS
 20  READ  X$
 30  IF  X$="%"  THEN  PRINT:
     GOTO  20
 40  IF  X$="$"  THEN  END
 50  PRINT  X$;
 60  GOTO  20
 70  END
 80  DATA  ムカシ  ムカシ  ,アルト
           コロニ  ,オジイサント  ,
           オバアサンガ  ,
 90  DATA  スンデイマシタ ,%, オジ
           イサンハ  ヤマヘ  シバカ
           リニ  ,
100  DATA  オバアサンハ  ,カワヘ  ,
           センタクニ  ,イキマシタ。
           ,$
```

```
RUN
ムカシ ムカシ アルトコロニ オジイサント オバアサン
ガ スンデイマシタ
オジイサンハ ヤマヘ シバカリニ オバアサンハ カワ
ヘ センタクニ イキマシタ。
Ok
■
```

● RESTORE(リストア)

READ～DATA 命令によって読み込まれるデータは、一度読まれるともう二度と読み込まれることはありません。でも、同じデータを何度も使いたいということはよくあることです。
そのために、一度読み終ったデータをもう一度読み込む命令があります。それが RESTORE 命令です。

```
RESTORE 行番号
```

とすると、次からは、「行番号」の場所にあるデータから読み込みを始めます。

```
10  READ  A, B, C
20  RESTORE  70
30  READ  A$, B$
40  PRINT  A+B+C ; A$+B$
50  END
60  DATA  121
70  DATA  202, 57
```

上のプログラムは、まず行番号10の READ 命令で、行番号60，70にある121、202、57の数を読みます。次に RESTORE 命令で行番号70からを指定しています。従って行番号30の READ 命令でA $には「202」、B $には「57」がはいることになります。 さっそく RUN させてみましょう。

```
RUN
 380 20257
Ok
■
```

行番号20の RESTORE 70 を取ってしまったらこのプログラムはどうなるでしょうか。

```
20   RETURN
RUN  RETURN
```

としてみましょう。行番号30の READ 命令では、もう読み込むべきデータが残っていません。 READ～DATA 命令で、読み込むデータの数が足りないとエラーになってしまいます。

```
20
RUN
Out of DATA in 30
Ok
■
```

READ A$, B$, C$…いくつかの文字変数
↕
DATA わたしは, MB-H3 です…対応する文字
( ) " " 「 」で
かこまなくてもよい。

〈起こりやすいエラー〉
● Out of DATA(アウト オブ データ)

READ～DATA 命令で、データの数が足りない場合に起こります。264ページをお読みください。

漢字を使ってプログラミング

(PUT KANJI)

これまで、ベーシックの中では、英数字、カタカナ、ひらがなを使ってきましたが、別売の漢字ROMを使うと、プログラムの中で漢字を使うことができます。
本機のグラフィック文字の中にも「月火水木…」などいくつかの漢字が用意されていますが、いずれもカタカナなどと同じ大きさで、少し見づらいですね。
漢字ROMには、漢字一文字が縦16ドット×横16ドットの大きさで、3000文字位のかな文字や漢字、特殊な記号が記憶されています。これらの漢字はキャラクターコードと同じように漢字コードという番号がついています。
本機で、漢字を表示させるときには、この漢字コードを使います。どの漢字が何番のコードを持っているかは、漢字ROMについているJIS漢字コード表などをご覧ください。
また、漢字はこれまでの文字と違って、SCREEN 5から8までのグラフィック画面に表示されます。
漢字を表示するためには、まずSCREEN命令を実行しておかなければなりません。

●PUT KANJI (プット カンジ)
グラフィック画面に漢字を表示させるための命令が、PUT KANJIです。
この命令では漢字を表示する位置を書き込まなければなりませんが、この位置は一つの漢字の左上すみの画面位置を（縦方向の位置，横方向の位置）で書きます。

この位置を書きます

漢字

PUT KANJI（漢字の左上すみの位置），
漢字コード，カラーコード

目的の漢字のコード

どんな色で漢字を書くかを決めます

さっそくためして見ることにしましょう。

漢字ROMをお持ちの方は、さっそく本機の電源をいったん切り、ROMカートリッジをカートリッジスロットに差し込んでください。
電源を再び入れれば準備OKです。

次のプログラムを打ち込んでください。

```
10  SCREEN 5
20  X=48
30  FOR I=1  TO 5
40  READ K$:K=VAL ("&
    H"+K$)
50  PUT KANJI (X, 88), K, 15
60  X=X+16
70  NEXT I
80  IF INKEY$="" THEN
    GOTO 80
90  END
100 DATA 372F, 244E, 4342, 4038,
    467C
```

RUNさせてみましょう。

君 の 誕 生 日

このプログラムを終了させるときには、なにかひとつキーを押してください。
(漢字ROMがない場合には何も表示されません。)

行番号100には漢字コード表から16進数で漢字コードを書いてありますので、この漢字コードを目的の漢字コードに入れ替えれば好きな漢字を表示させることができますね。

プリンタで漢字を印刷したい場合に注意しなければならないのは、LPRINT命令では漢字が印刷できないということです。
さきほどのプログラムの実行結果をプリンタで印刷するときには、ハードコピーをとる方法があります。
コンピュータ本体にハードコピー命令が内蔵されている場合（例えばCALL HCOPYなど）には、次の命令を追加しておくと、プリンタで漢字が印刷できます。

```
75 CALL HCOPY
```

本格的なワードプロセッサのようなわけにはいきませんが、漢字が使えることでコンピュータの用途が広がります。
プログラムの組みかたしだいで、漢字を画面で選んだり好きな位置に表示させることも楽にできるようになります。
漢字ROMはとても便利です。機会があれば、ぜひ用意してください。

中身をのぞいてみよう

(PEEK関数，POKE，CLEAR)

さあ、ベーシックの長い道のりもあとわずかで終了です。ベーシックというのはどんなものなのか、理解できたでしょうか？スプライトを使って絵を動かしたり、PLAY文で音楽を演奏したり、なかなか楽しいことができるでしょう。

ここでは、ちょっとむずかしいかもしれませんがコンピュータの中身をのぞいてみるお話をします。本機は、こんなこともできるのかという程度で読んでみてください。

●メモリって何？

キーボードから文字や数字を打ち込んでも、画面がいっぱいになれば、どんどん消されてしまいました。しかし、先頭に行番号をつけてプログラムにしてしまえば、それをいつまでも(NEWとするか電源を切るまで)覚えているのでしたね。

いったいプログラムをどこに覚えておくのでしょう。じつはベーシックで書かれたプログラムを覚えておいたり、ベーシックを動かすためのいろいろな命令をしまっておくための場所があるのです。それをメモリ(記憶装置) といいます。

メモリは使いやすいように1つずつ0から順番に番号がふってあります。本機はメモリの0から65535 (&HFFFF) まで使うことができるのです。このメモリにふった番号を番地と呼んでいます。

本機はメモリをどんなふうに使っているのでしょうか？

番地ごとに、ちょっと説明してみましょう。

16進	内容	10進
&HFFFF	変数など	65535
&HF380	プログラム	62336
&H8000	ベーシックを動かすための命令	32768
&H0000		0

イ） 0～32767(&H7FFF)番地

ベーシックを動かすための命令をしまっておく場所です。ここで、プログラム中にあるさまざまな命令を理解して、実行していくのです。この部分は変化しません。

ロ） 32768(&H8000)～65535(&HFFFF)番地

プログラムや変数をしまっておくための場所です。プログラムの作成や、実行にともなって、この部分のメモリの内容は変化します。

メモリの使われ方はだいたい理解できましたか？では実際にメモリに何が書いてあるのか、のぞいてみることにしましょう。

●PEEK（ピーク）関数

メモリの中をのぞくときに使うのがこのPEEK関数です。

PEEK(番地)

とすると、その番地のメモリの内容を値とするのです。さっそく使ってみましょう。

PRINT PEEK（0）RETURN

としてみましょう。

```
Ok
PRINT PEEK(0)
 243
Ok
█
```

どんな値が画面に書かれましたか？ 243という数字が、メモリの0番地にはいっていますね。かっこの中の数字をいろいろに換えて、もっと他の番地ものぞいてみましょう。

えっ？なになに、数字しか出てこないのでおもしろくないって？　それは当然なのです。0番地から32768番地まではベーシックを動かすための命令がはいっているのですが、そもそもコンピュータは命令を数字で表現して覚えているのです。数字がどんな命令に当たるのかを知りたい人は機械語の勉強をすることをおすすめします。

●POKE, CLEAR（ポーク、クリア）

今度はメモリに好きな数字を書き込んでみましょう。そんなときに使うのがPOKE命令です。

POKE 番地,データ

とすれば、その番地のメモリにデータを書き込めます。この命令を使う前に、もう一度さきほどの図を見てください。メモリにかってに変な値を書き込んでもだいじょうぶなのでしょうか？ちょっと心配ですね。メモリの中に、かってに使ってもかまわない場所をとることはできないのでしょうか？

そんな時にはCLEAR命令を使います。CLEAR命令はベーシックが使うメモリの上限を決める命令です。たとえば

CLEAR 300, &HDFFF

とすると、ベーシックは＆HDFFF番地までしかメモリを使用せず＆HE000～＆HF37Fの4992個分が、自由に使えるようになるわけです。

CLEAR　文字領域の大きさ，番地

CLEAR 300, &HDFFF実行後

さて、それではPOKE命令でメモリに何か書き込んでみましょう。

```
POKE & HE001,100 RETURN
```

としてください。本当に書き込むことができたでしょうか？PEEK関数を使って確かめてみましょう。

```
PRINT PEEK(&HE001) RETURN
```

だいじょうぶ、ちゃんと書かれていましたね。

```
Ok
POKE &HE001,100
Ok
PRINT PEEK(&HE001)
 100
Ok
■
```

PEEK関数やPOKE命令を使うと、コンピュータの世界はさらに広がりますが、そのためには機械語の知識が必要です。機会があればぜひ機械語も勉強してください。

〈起こりやすいエラー〉

●Illegal function call(イリーガル ファンクションコール)

POKE 命令で書き込むデータが0～255の範囲にない場合に起こります。たとえば、

```
POKE &HD200,500
```

などとした場合です。

●Overflow(オーバーフロー)

PEEK,POKE,CLEAR で指定する番地が0～65535の範囲になかった場合に起こります。たとえば、

```
PEEK (65536)
```

などとした場合です。

●Out of memory(アウト オブ メモリー)

CLEAR命令で、ベーシックで使うメモリの上限を小さくとりすぎた場合です。もっと大きな値にするか、プログラムを短くするかしてください。

ベーシック文法編

「ベーシック基礎編」を理解して、自分でいろいろなプログラムを作りはじめると、もっと楽しく、もっといろいろなことに使えそうだと感じることがあると思います。そんなときに、便利なコマンドやステートメント、関数を知っていると、意外に簡単な方法で、思ったとおりのことをさせることができたりします。
ベーシック文法編では、本機で使うことのできる命令や、関数を、機能別に分けて1つずつ説明します。この本の最後にアルファベット順の索引もありますので　プログラム作成時の辞書がわりに使うと便利です。

はじめに

ベーシック文法編では、本機で使用できるベーシックのコマンド、ステートメント、関数をすべて説明します。
このベーシック文法編は、ベーシック基礎編と違い、全文を読んでいきながら動作を確認していくというよりも、皆さんが実際に目的を持ったプログラムを作成していくときに必要となる関数やコマンドなどを調べるのに便利なようになっています。つまりベーシック基礎編を教科書とすると、ベーシック文法編は辞書ということになります。
ベーシック文法編には、基礎編で説明していないコマンドやステートメント、関数が多く出てきますが、新しいものに積極的にチャレンジして、どんどん自分のものにしていきましょう。
ベーシック文法編は、機能別に分類して説明してありますが辞書として使用していただくのに便利なようにこの本の最後にＡＢＣ順（アルファベット順）の索引もあります。

MSXベーシックの概要

1．行の形式

ｎｎｎｎ　ベーシックのステートメント［：ベーシックのステートメント……］

●ｎｎｎｎは行番号です。

●１行には、ベーシックのステートメントを「：」(コロン)で結ぶことにより、複数のベーシックのステートメントを書くことができます。

●１行には、255文字まで書くことができます。

2．行番号

●0から65529までの整数を使用します。

●LIST、AUTO、DELETE命令で「．」(ピリオド)を使用すると、現在の行番号を参照します。

3．使用可能な文字

●英大文字、英小文字、ひらがな、カタカナ、数字(0～9)、英記号、かな記号、グラフィック文字が使用できます。それぞれの内容は、付録の263ページをご覧ください。

4．定数

●数値定数と文字定数とがあります。

●**文字定数**

最大255文字までの文字列で「"」（ダブルクォート）ではさんで使用します。

例． "ABCD"

"230B"

●**数値定数**

数値の表現の形式で次の５種類にわけられます。

i）**固定小数点定数**

通常用いている正負の実数です。

例． 127

−703

12.76

ii）**浮動小数点定数**

◦指数形式で表わされる正負の数値です。

次の形式で表わされます。

±ｎ…ｎ．ｎ…ｎＥ±ｎｎ

または

±ｎ…ｎ．ｎ…ｎＤ±ｎｎ

±ｎ…ｎ．ｎ…ｎ ── 固定小数点定数部

Ｄ±ｎｎ ── 指数部(10^{-64}～10^{63})

例． 235.488Ｅ−7　（＝0.0000235488）

−611Ｄ＋5　（＝−61100000）

iii）**２進定数**

◦２進法で表わされた数値です。

◦数値の前に「＆Ｂ」をつけます。

例． &B10010　（＝18）

iv）**８進定数**

◦８進法で表わされた数値です。

◦数値の前に「＆Ｏ」をつけます。

例． &O777　（＝511）

v）**16進定数**

◦16進法で表わされた数値です。

◦数値の前に「＆Ｈ」をつけます。

例． &H7FFF　（＝32767）

●**数値の型について**

内部の表現形式で次の３種類に分けられます。

i）**整数型**

◦数値の後に「％」(パーセント)をつけます。

◦−32768～32767の範囲の整数です。

例． −3000％

27％

ii）**単精度型**

◦数値の後に「！」(感嘆符)　をつけます。

◦６桁まで記憶、表示されます。(７桁目を四捨五入します。)

。また「E」を使った指数形式もこの型になります。

例. −2.13E−13
26.5!

iii）**倍精度型**

。数値の後に「#」(シャープ)をつけます。

。14桁まで記憶、表示されます。(15桁目を四捨五入します。)

。また、「D」を使った指数形式もこの型になります。

例. −611D+5

●特に指定を行わなければ、倍精度で記憶，計算，表示を行います。

定数のまとめ

定数	表現形式	型	数値例
数値定数	固定小数点	整数型	12%，−32000%
		単精度型	1.25!，57000!
		倍精度型	1.2，−256#
	浮動小数点	単精度型	1.0E+7
		倍精度型	−2.56D−21
	2進数	整数型	&B10011
	8進数	整数型	&O2777
	16進数	整数型	&HABFF
文字定数	文字列	文字型	"ABCD"，"−23AF"

5．変数

●変数名の長さは、255文字まで可能ですが、3文字目以降の文字は識別されません。

●変数名には、英文字、数値を使用しますが、先頭の1文字目は必ず英文字でなければなりません。

●ベーシックの命令、関数は変数名に使用できません。また、命令、関数を変数名の中に含んでもいけません。

。詳細は基礎編27ページをご覧ください。

●変数には、文字変数と数値変数があります。

●文字変数には、変数名の後に「$」(ドル記号) をつけ、255文字まで入れることができます。

●数値変数にはつぎの3種類があります。

i）整数型変数……変数名の後に「%」(パーセント)記号をつけます。

ii）単精度型変数…変数名の後に「!」(感嘆符)記号をつけます。

iii）倍精度型変数…変数名の後に何もつけないか、または「#」(シャープ)をつけます。

●DEFINT, DEFSNG, DEFDBL, DEFSTR を使用すると、それぞれ変数の型を宣言することができます。詳細は168ページをご覧ください。

●変数を記憶するメモリの大きさはつぎのとおりです。

整数型変数	2バイト
単精度型変数	4バイト
倍精度型変数	8バイト
文字変数	入れる文字数＋3バイト

6．型変換

●数値定数を文字変数に、またはその逆に変換する場合には、関数STR$，VALを用います。

●ある型の数値(定数または変数)を他の型に変換する場合には、その型の数値を、希望する型の数値変数に代入します。

●式の計算は、演算数値中の一番高い精度に変換されてから行なわれます。ただし、返される数値は、指定がある場合には指定された精度になります。

例．

```
10   D=1／3!
20   PRINT D
RUN
.3333333333333333
```

演算は倍精度で行なわれます。結果は倍精度で表示されます。

```
10   D!=1／3
20   PRINT D!
RUN
.333333
```

演算は倍精度で行なわれます。結果は単精度で表示されます。

●論理関係式の演算数は、整数に変換され、結果は整数で返されます。演算数は、－32768から、32767の範囲の数値で、それ以外は、Over flowエラーとなります。

●浮動小数点(実)数値を整数に変換すると、小数部分は、切り捨てられます。

●単精度型を倍精度型変数に変換する場合には、上位６桁のみが有効となります。これは、単精度型数値が６桁の精度しかもっていないからです。

7．式

●文字または数値の定数、変数あるいはそれらの組合せを、演算子を用いて一つの値を作りだすものを式といいます。

●式には、算術式、関係式、論理式、関数の４種類があります。

●算術演算子

つぎの演算子を用います。

		例
＋	加算	X＋Y
－	減算	X－Y
＊	乗算	X＊Y
／	除算	X／Y
^	べき乗	X^Y
－	負数	－X
¥	整数除算	X¥Y(10¥4＝2)
MOD	整数剰余	XMODY (10.4MOD4＝2)

●比較演算子

ｉ）２つの値の比較に使います。

ii）つぎの演算子を用います。

		例
＝	等しい	X＝Y
<>	等しくない	X<>Y
<	未満	X<Y
>	超	X>Y
<＝	以下	X<＝Y
>＝	以上	X>＝Y

iii）１つの式の中に、算術式と関係式の両方がある場合には、算術式が評価された後に、関係式が評価されます。

●論理演算子

ｉ）論理演算子は以下の６種類があり、整数型の数値に適用できます。数値の各ビットについて以下の演算を行うことに注意してください。

演算子	使用法	評価
NOT (ノット)	NOT X	Xではない X ／ NOT X 0 ／ 1 1 ／ 0
AND (アンド)	X AND Y	Xであり、かつYである X ／ Y ／ X AND Y 0 ／ 0 ／ 0 0 ／ 1 ／ 0 1 ／ 0 ／ 0 1 ／ 1 ／ 1
OR (オア)	X OR Y	Xまたは、Yである X ／ Y ／ X OR Y 0 ／ 0 ／ 0 0 ／ 1 ／ 1 1 ／ 0 ／ 1 1 ／ 1 ／ 1

演算子	使用法	評　価
XOR (イクスクルーシブ・オア)	X XOR Y	XとYは等しくない X | Y | X XOR Y 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
EQV (イクイバレント)	XEQVY	XとYは等しい X | Y | XEQVY 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1
IMP (インプライ)	XIMPY	XはYを含む X | Y | XIMPY 0 | 0 | 1 0 | 1 | 1 1 | 0 | 0 1 | 1 | 1

例1．NOT−2　………　1

（　　−2＝&B1111111111111110
NOT−2＝&B0000000000000001＝1）

例2．23AND7　………　7

（　　23＝&B10111
AND 7＝&B00111
　　&B00111＝7）

例3．8OR6　………　14

（　　8＝&B1000
OR　6＝&B0110
　　&B1110＝14）

例4．16 XOR6　………　22

（　　16＝&B10000
XOR 6＝&B00110
　　&B10110＝22）

例5．7 EQV 8　………−16

（　　7＝&B0000000000000111
EQV 8＝&B0000000000001000
　　&B1111111111110000＝−16）

例6．12 IMP 5　………−9

（　　12＝&B0000000000001100
IMP　5＝&B0000000000000101
　　&B1111111111110111＝−9）

ii）論理演算子は演算子の中で最も低い優先度となります。

●関数

i）与えられた演算数値をもとに、あらかじめ決められた演算を行ない、その結果の値を返すものが関数です。

ii）プログラム中で、ユーザが設定した関数を使うこともできます。(DEF FN関数)

●文字列の演算

i）文字列は「＋」記号を使って、つなぐことができます。

ii）数値の比較演算子を用いて、文字列の比較を行なうことができます。比較の結果が真ならば−1，偽ならば0となります。

iii）文字列の比較演算は、文字列の先頭から一文字ずつキャラクターコードを比較して行なわれます。もし比較の途中でキャラクターコードが異なった場合は、小さいコードを含む文字列が小さいとされます。また一方の文字列が終りになった場合には文字列の短い方が、小さいとされます。

iv）「␣」(空白)も比較の対象になります。

例．"MSX"＋"ベーシック"＝"MSXベーシック"
"RED"＜＞"BLACK"
"RECORD"＜"RECORDER"
"IN PUT"＞"IN ␣ PUT"

8．エラー

●プログラム実行中、エラーが発生した場合、画面にはエラーメッセージが表示されます。（付録264ページ）

●エラーが発生した時点でプログラムの実行は中断され、コマンド待ちの状態になります。

●中断されたままの状態では、変数の値、プログラムは残っています。

●プログラム中のエラーは、その行が実行されるまで検出されません。

MSXベーシックとMSXベーシックver 2との互換性について

従来のMSX規格で作られたソフトウェアはすべて使用可能です。

MSXベーシックver 2で作成したソフトウェアは、下記の命令やステートメントなどを使用していないものについてはそのまま使用可能です。

●MSXベーシックver 2で拡張または追加された命令、関数

- **SCREEN命令のうち**
  SCREEN　4～8までの画面モードこれにともなうグラフィックス関係の命令のうちSCREEN 4～8の画面モードに対応した座標値を持つもの、画面との論理演算を行うもの、ならびに色指定をカラーパレットで行ったもの
  PSET、PRESET、LINE、CIRCLE
- **WIDTH**
- **COLOR命令のうち**
  パレット機能を使用したもの
  書式　COLOR＝を使用したもの
- **COPY**
- **COLOR SPRITE**
- **COLOR SPRITE$**
- **BASE VDP命令のうち**
  新設されたレジスタにアクセスするもの
- **VPEEK VPOKE**
- **SET命令による拡張命令**
  SET ADJUST
  SET BEEP
  SET DATE
  SET TIME
  SET NAME
  SET TITLE
  SET PASSWORD
  SET PROMPT
  SET SCREEN
  SET VIDEO
- **CALL命令による拡張命令**
  CALL MEMINI
  CALL MFILES
  CALL MKILL
  CALL MNAME
  CALL SPAGE
- **GET TIME**
- **GET DATE**
- **PUT KANJI**
- **PAD**
  ライトペン、マウスからの入力を取り込む場合
- **BSAVE, BLOADのうち**
  VRAMの内容を直接SAVE, LOADするもの

  書式　BSAVE　“ファイル名”, S
  　　　BLOAD　“ファイル名”, S

10. メモリーディスクの機能について

MSXベーシックver 2では、RAMのうち&H0000から&H 7 FFFの間の番地をRAMディスクとして使用することが可能です。
使用する場合には、CALL MEMIMIを実行し、初期化を行わなければなりません。その後は通常のディスクと同じ扱いが可能になります。ただし、電源を切るとSAVEされていた内容は消去されます。また、再び電源を入れた場合には、その度に初期化を行わなければなりません。

RAMディスクは、ファイルディスクリプタとして、“MEM：ファイル名”を使用します。

ファイル操作にあたっては、ディスクベーシックの命令がそのまま使用できますが、ランダムアクセスファイルは扱うことができません。

11．文法編の使い方

●命令、ステートメント、関数は、それぞれ機能別（理解しやすい順序）になっています。辞書のようにして目的の命令語をさがすときは、267ページの索引をお使いください。

●説明文中の書式について

i）アルファベットの大文字で示されている項目はそのまま入力します。

ii）カギカッコ〈 〉で囲まれた項目は、あなたが指定します。

iii）角カッコ[ ]で囲まれた項目は、省略することができます。

iv）省略記号……の続く項目は、1行の許す範囲内で任意の回数を繰り返すことができます。

v）｜ ; または , ｜のように書かれたものは、上下どちらを使っても、よいことを意味します。

vi）上記以外の記号は、文法的に必要な記号ですから、プログラム中に書かれなければなりません。

vii）解説などで出てくる「＆H」に続く数値は16進数であることを示します。

viii）「－」は、マイナス記号を示します。

コマンド

LIST
（リスト）

➡ 基礎編30ページ

機能　**プログラムリストを画面に出力します。**

書式　LIST［〈始点行番号〉］［－［〈終点行番号〉］］

文例　LIST　100－300
（行番号100から300までのプログラムを画面に表示します。）

解説

現在メモリ上にあるベーシックプログラムの一部、または全部を画面に表示します。２つの行番号を省略するとプログラム全部を、〈始点行番号〉を省略するとプログラムの最初から〈終点行番号〉まで、〈終点行番号〉を省略すると〈始点行番号〉以降プログラムの最後まで出力します。LISTコマンドの後に「．」（ピリオド）だけを入力すると、現在実行が停止している行を表示します。

LLIST
（エル リスト）

➡ 基礎編37ページ

機能　**プログラムリストをプリンタに出力します。**

書式　LLIST［〈**始点行番号**〉］［**－**［〈**終点行番号**〉］］

文例　**LLIST　100－300**
（行番号100から300までのプログラムをプリンタに出力します。）

解説

現在メモリ上にあるベーシックプログラムの一部、または全部をプリンタに出力します。リストする行番号の指定はLISTの場合と同じです。

AUTO
（オート）

➡ 基礎編39ページ

機能　**行番号を自動的に発生させます。**

書式　AUTO　［〈**行番号**〉］［**,**〈**増分**〉］

文例　**AUTO　100，5**
（行番号を100、105、110…の順に発生させます。）
AUTO　，5　とすると、行番号を0、5、15…の順に発生させます。

解説

このコマンドを実行すると、RETURN キーを入力するたびに、〈行番号〉に始まり〈増分〉ずつ増加していく行番号を自動的に発生させます。AUTOコマンドの後に「．」（ピリオド）を入力すると、現在実行が停止している行から、行番号を発生します。

注） すでに使用されている行番号を、AUTOで発生させた場合には、行番号のすぐ後に「＊」が表示されて、プログラムが変更されることを警告します。このとき、RETURN だけを押すとその行は保護されます。

RENUM
(リナンバー)

➡ 基礎編40ページ

機能　**プログラムの行番号をつけ直します。**

書式　**RENUM［〈新行番号〉］［，〈旧行番号〉］［，〈増分〉］**

文例　**RENUM　200，100，20**

(行番号100を、200とつけなおし、以後20番飛びに行番号をつけかえます。)

解説

〈旧行番号〉で指定する行より後ろの行番号を、〈新行番号〉から始まり〈増分〉ずつ増えていく新しい行番号につけ直します。これらを省略した場合は次のとおり設定されます。

〈新行番号〉 10

〈旧行番号〉 現在あるプログラムの最も小さい行番号

〈増　　分〉 10

また、この命令で、GOTO、GOSUB、IF～THEN、ON～GOTO、ON～GOSUB、ERLの中にある行番号もつけ直します。ただしこれらの行番号が、プログラムの中に存在していない場合、「Undefined line number」とエラー表示して、その行番号はそのまま残ります。

行番号の順序が変化するようなパラメータを与えると、Illegal function call エラーが出ます。

DELETE
(デリート)

➡ 基礎編41ページ

機能　**指定した行を消します。**

書式　**①DELETE 〈始点行番号〉［〈－終点行番号〉］**

②DELETE －〈終点行番号〉

文例　**DELETE　50－150**

(行番号50から150までを消します。)

DELETE　－300

(最初の行から、行番号300までを消します。)

解説

<始点行番号>から<終点行番号>までのプログラムを消します。<始点行番号>のみを指定した場合はその行1行のみを消します。書式②の場合はプログラムの最初からその行までを消します。
DELETEコマンドの後に「.」(ピリオド)だけを入力すると、現在実行が停止している行を消します。<始点行番号>がプログラム中に存在しない場合は、次の行番号から実行します。また<終点行番号>がない場合には Illegal function call エラーが出ます。

NEW
(ニュー)

➡ 基礎編28ページ

機能　**プログラムを消し、変数をクリアします。**

書式　NEW

解説

メモリ上のベーシックプログラムを消し、すべての数値変数を0に、文字変数を「" "」(ヌルストリング)にします。プログラム中にこの命令が現われると、実行をやめプログラムを消し、変数をクリアして、命令待ちの状態になります。開かれているファイルは、閉じられます。

RUN
(ラン)

➡ 基礎編29ページ

機能　**プログラムを指定された行から実行します。**

書式　RUN [<行番号>]

文例　RUN　100
(行番号100の命令から実行します。)

解説

プログラムを<行番号>から実行します。<行番号>を省略した場合、プログラムの先頭から実行を始めます。
プログラムの実行に先だって、開かれているファイルを閉じます。

CONT
(コンティニュー)

➡ 基礎編43ページ

機能　**中断したプログラムの実行を再開します。**

書式　CONT

解説

[CTRL]＋[STOP]入力後、またはSTOP文、END文を実行後、コマンド待ちの状態でCONTを入力すると、停止した次のステートメントから実行を再開します。INPUT文で停止した場合は、そのINPUT文から実行を再開します。ただし、停止後、プログラムの編集(追加、訂正、削除)を行った場合、CONT命令によって実行を再開することはできません。

TRON/TROFF
(トレース オン/トレースオフ)

機能　**トレースモードの設定・解除を行ないます。**

書式　TRONまたはTROFF

解説

①TRON

TRON命令が実行されると、ベーシックはトレースモードに入ります。このモードでは、実行したプログラムの行番号を「[」、「]」でくくって画面に表示します。おもにプログラムのデバッギングに使用します。

②TROFF

トレースモードを解除します。また、NEW命令を実行しても同じ結果が得られます。

サンプルプログラム

```
10  TRON
20  FOR N=1  TO  10
30  PRINT  N
40  NEXT  N
50  TROFF
60  END
```

(行番号20から50までのトレースを行ないます。)

CSAVE
(シー セーブ)

➡ 基礎編33ページ

機能 **カセットテープにBASICのプログラムを記録します。**

書式 **CSAVE "<ファイル名>" [, <ボーレイト>]**

文例 **CSAVE "HINT" , 2**

(プログラムを "HINT" というファイル名で、カセットテープに2400ボーで記録します。)

解説

ベーシックエリアにあるプログラムに<ファイル名>をつけてカセットテープにセーブします。

<ボーレイト>はカセットレコーダへの書き込み速度を指定します。1、2の2種類が指定できます。

1：1200ボー

2：2400ボー

<ボーレイト>が省略された場合、SCREEN命令で指定された速度で書き込みを行います。

プログラムは短縮された中間言語の形で記録されます。

CLOAD
(シー ロード)

➡ 基礎編35ページ

機能 **カセットテープからメモリへプログラムをロードします。**

書式 **CLOAD ["<ファイル名>"]**

文例 **CLOAD "HINT"**

(カセットテープから "HINT" というファイル名のプログラムを読み込みます。)

解説

カセットテープから<ファイル名>で指定したプログラムを読み込みます。<ファイル名>を省略すると、テープ上で最初に見つけたプログラムをロードします。ボーレイトは自動的に決定されます。

CLOAD?
(シー ロード ベリファイ)

➡ 基礎編35ページ

機能　**カセットテープにセーブしたプログラムの内容とメモリにあるプログラムの内容を比較します。**

書式　**CLOAD? ["<ファイル名>"]**

文例　**CLOAD? "HINT"**

(カセットテープにセーブした" HINT" というファイル名のプログラムが、メモリ内のプログラムの内容と同じかどうか、確認します。)

解説

カセットテープのプログラムの内容とメモリにあるプログラムの内容が同じであれば Ok と表示し、もしそうでなければVerify error とエラー表示されます。

LOAD
(ロード)

機能　**プログラムをファイルから読み込みます。**

書式　**LOAD "<デバイスディスクリプタ> [<ファイル名>]" [, R]**

文例　**LOAD "CAS : SAMPLE", R**

(カセットから "SAMPLE" というファイル名のプログラムを読み込み、ただちに実行します。)

解説

<デバイスディスクリプタ>で指定した装置から、<ファイル名>のプログラムファイルをメモリに読み込みます。(デバイスディスクリプタについてはOPEN命令を参照してください。)

この命令を実行すると、すべてのファイルは閉じられ、すでにメモリに存在していたプログラムは消されます。ただしあとにRをつけた場合、ファイルは閉じられることなく、またプログラムを読み込んだ後、ただちにそのプログラムを実行します。

もし、<ファイル名>を省略すると、最初に現れたプログラムを読み込みます。

SAVE
(セーブ)

機能　**メモリ内にあるベーシックのプログラムを指定した装置に記録します。**

書式　**SAVE "〈デバイスディスクリプタ〉[〈ファイル名〉]"**

文例　**SAVE "CAS:SAMPLE"**

（カセットテープレコーダに「SAMPLE」という名前のプログラムを記録します。）

解説

〈デバイスディスクリプタ〉で指定した装置に、〈ファイル名〉をつけて、ベーシックのプログラムを記録します。

CTRL ＋ Z (&H1A)が、ファイルの終了として記入されます。

プログラムはアスキー形式（文字形式）で記録されます。

BLOAD
(ビー ロード)

機能　**機械語プログラムをファイルから読み込みます。**

書式　**BLOAD"〈デバイスディスクリプタ〉[〈ファイル名〉]"[,RまたはS]**
[,〈オフセット〉]

文例　**BLOAD" CAS：マシンゴ", R**

（カセットから"マシンゴ"というファイル名の機械語プログラムを読み込み、ただちに実行します。）

解説

ファイルから機械語プログラムをメモリ上に読み込みます。〈ファイル名〉を省略すると最初に現れたプログラムを読み込みます。

〈オフセット〉を省略すると機械語プログラムはBSAVEでセーブするときに指定した〈開始アドレス〉からロードされますが、〈オフセット〉を指定するとセーブの際に指定された開始アドレスに、〈オフセット〉を加えた番地にロードします。したがって、〈オフセット〉を付けてロードするプログラムは、セーブされたときと異なる番地にロードされても、実行可能（リロケータブル）なプログラムでなければなりません。Rをつけるとプログラムの読み込みが終った後、BSAVEで指定しておいた実行開始番地からただちに実行開始します。

ただし、〈オフセット〉を指定してロードした場合には実行開始はできません。

文例　BLOAD"CAS：マシンゴ", R, &H1000

またSをつけるとVRAMに内容を読み込みます。画面モードが5～8の場合には、アクティブページに内容が読み込まれます。

BSAVE
(ビー セーブ)

機能　**メモリ上にある機械語プログラムをファイルに記録します。**

書式　BSAVE"〈デバイスディスクリプタ〉［〈ファイル名〉］",〈先頭番地〉,〈終了番地〉［,S］［,〈実行開始番地〉］

文例　BSAVE"CAS：マシンゴ"&HA000,&HAFFF

(&HA000から&HAFFFまでの機械語プログラムに"マシンゴ"というファイル名をつけてカセットに記録します。)

解説

メモリ上にある機械語プログラムをファイルに書き込みます。メモリのどこからどこまでをファイルに書き込むかを指定するために〈先頭番地〉と、〈終了番地〉を指定します。
〈実行開始番地〉はBLOAD命令の「R」オプションで読み込んだ機械語プログラム実行させるときの、スタート番地を記入します。
また〈実行開始番地〉を省略すると〈先頭番地〉が〈実行開始番地〉とみなします。
プログラムはバイナリ形式で記憶されます。
また、最後にSをつけるとVRAMの内容が書き込まれます。
画面モードが5～8の場合には、アクティブページの内容が書き込まれます。

MERGE
(マージ)

機能　**メモリ上にあるプログラムとカセットテープに記録されているプログラムとを合わせます。**

書式　MERGE "〈デバイスディスクリプタ〉［〈ファイル名〉］"

文例　MERGE "CAS：SAMPLE"

(カセットから "SAMPLE" というファイル名のプログラムを読み込んで、メモリ上にあるプログラムと合わせます。)

解説

現在メモリ上にあるベーシックプログラムと、カセットテープ上の指定したファイル名のプログラムとを合わせてひとつのプログラムにします。〈ファイル名〉を省いたときには最初にみつけたものを読み込みます。
プログラム中にMERGE命令があると、実行後、命令待ちに戻ります。

注）MERGEに使うプログラムファイルはSAVEコマンドで記録されたものでなければなりません。メモリ上のプログラムとカセットテープ上のプログラムに同じ行番号があれば、その行番号の内容はテープ上のプログラムの内容に置き換わります。

CLEAR
（クリア）

➡ 基礎編135ページ

機能　**変数値をクリアし、文字領域の大きさとベーシックで使用するメモリの上限を設定します。**

書式　**CLEAR［〈文字領域の大きさ〉［,〈ベーシックで使用するメモリ番地の上限〉］］**

文例　**CLEAR 300 , &HDFFF**

(文字領域の大きさは、300バイト、ベーシックで使用するメモリ番地の上限は、&HDFFFと設定します。)

解説

すべての数値変数を0に、文字変数を「" "」(ヌルストリング) に初期化します。またこのとき開かれていたファイルは全て閉じられます。
パラメータを指定することによって文字領域の大きさと、ベーシックで使用するメモリ番地の上限を設定できます。〈ベーシックで使用するメモリの上限〉を指定する場合、〈文字領域の大きさ〉を省略することはできません。
電源投入時には〈文字領域の大きさ〉は200バイトに設定されています。
なお、使用できるメモリ番地の上限は&HF380です。

CLEAR300, &HDFFF実行後のアドレスマップ

アドレス	領域
FFFFH	ワークエリア
F380H	マシン語領域
E000H	ベーシックプログラム領域
8000H	

一般ステートメント

LET
(レット)

機能　変数に式の値を代入します。

書式　[LET]〈変数名〉=〈式〉

文例　LET　A=5
(変数Aに5を代入します。)

解説
〈式〉で表される値を変数や配列に代入します。この等号は代入を表わすものです。「LET」は省略してもかまいません。
変数の形と〈式〉の値の形(数値か文字か)が一致していなければなりません。〈式〉は文字式の場合もあります。

サンプルプログラム

```
10  LET A=5
20  LET B$= "SAMPLE"
30  PRINT A, B$
40  END
```

(変数AとB$の値を画面表示します。)

REM
(リマーク)

➡ 基礎編41ページ

機能　プログラムに注釈を入れます。

書式　REM [〈注釈〉]

文例　REM これはけいさんプログラムです。

解説

この文から、行の終わりまでは、注釈とみなされ、実行時には無視されます。
REM文の後に「：」(コロン)で区切って実行文を書いても実行されません。

省略形は「’」(アポストロフィ)ですが、省略形で入力した場合には、リストの中でも「’」で表示され、「REM」に変換表示はされません。

FOR～NEXT
(フォー～ネクスト)

➡ 基礎編64ページ

機能　FORからNEXTの間のプログラムを指定した回数だけくり返し行ないます。

書式　FOR 〈変数名〉＝〈初期値〉 TO 〈終値〉 [〈STEP〈増分〉]
　　　⋮
　　　NEXT [〈変数名〉] [，〈変数名〉]…

文例　FOR N=0 TO 10 STEP2
　　　⋮
　　　NEXT N

(変数Nの値を0から2間隔で10まで変えながら、6回くり返します。)

解説

〈初期値〉、〈終値〉、〈増分〉には数式を用いることもできます。〈増分〉は負の値を用いることもできます。「STEP〈増分〉」を省いた場合は、増分は1となります。

FOR～NEXTは入れ子構造にすることができます。つまり、FOR～NEXTの中にまたFOR～NEXTを置くことができます。このとき1つのFOR～NEXTは完全に他のFOR～NEXTの中になければなりません。FOR～NEXTを重ね、終わる場所が同じであるときは、1つのNEXT文にまとめることができます。その時はNEXTに続けて、内側のFOR～NEXT文の〈変数名〉から順に、「,」(カンマ)で区切って書きます。

例
```
FOR N=1 TO 20
 :
FOR J=5 TO 10
 :
NEXT   J, N
```

また、NEXT文の〈変数名〉を省略することができます。その場合には、内側のFOR文の変数名とみなされます。

注) FOR文の実行時点で終了条件が成立していても、FOR～NEXTは、必ず1回は実行されます。

GOSUB～RETURN
(ゴーサブ～リターン)

➡ 基礎編70ページ

機能 サブルーチンの呼び出しと復帰を行ないます。

書式 GOSUB 〈行番号〉
RETURN [〈行番号〉]

文例
```
GOSUB  100
  :
RETURN  60
```
(行番号100から、始まるサブルーチンに飛び、サブルーチン終了後、行番号60に戻ります。)

解説

GOSUB文により〈行番号〉で示されたサブルーチンに飛び、終了後「RETURN」によりGOSUB文の次の文に戻ります。

RETURN文に〈行番号〉を指定すると、GOSUB文の次の文へは戻らず、指定した行に戻ります。

サブルーチン中で、CLEAR文が実行されると、RETURN文実行時、「RETURN without GOSUB」のエラーが出ます。

GOTO
(ゴートゥー)

➡ 基礎編42ページ

機能 指定した行番号へ無条件に飛びます。

書式 GOTO 〈行番号〉

文例　GOTO　120
（行番号120へ飛んで命令を実行します。）

解説
指定した〈行番号〉に無条件に実行が移ります。

IF～THEN（ELSE）
イフ～ゼン（エルス）

➡　基礎編60ページ

機能　論理式の値により実行すべき文を選択します。

書式　IF〈論理式〉THEN [|〈文〉[:〈文〉…] |]
　　　　　　　　　　　　　　|または〈行番号〉|
　　　　[ELSE |〈文〉[:〈文〉…] |]
　　　　　　　|または〈行番号〉|

文例　IF A$="Y" THEN PRINT "YES"
　　　　ELSE PRINT "NO"
（文字変数A$の値がYならYES、それ以外ならNOと画面に表示します。）

解説
〈論理式〉が真(成立する)ならば、THEN以下の文を実行し、ELSE文があるときはELSE文の前まで、ないときは行の最後までを実行します。もし〈論理式〉が偽(成立しない)ならば、ELSE文があるときはTHEN から ELSEまでの文は無視して、ELSE以下を実行します。ない場合には、次の行を実行します。
THENのあとにGOTO文がくる場合、THENまたはGOTOのどちらかを省略することができます。
また、ELSE後にGOTO文がくる場合には、GOTOを省略することができます。

ON GOTO/GOSUB
(オン ゴートゥー/ゴーサブ)

➡ 基礎編70ページ

機能　式の値により指定した行番号、あるいはサブルーチンへ飛びます。

書式　①ON 〈式〉 GOTO 〈行番号〉 [, 〈行番号〉…]
②ON 〈式〉 GOSUB 〈行番号〉 [, 〈行番号〉…]

文例　①ON A GOTO 70, 80, 90
(変数Aの値が1のときは行番号70、2のときは80、3のときは90へ飛びます。)
②ON A GOSUB 100, 200
(変数Aの値が1のときは、行番号100のサブルーチンへ、2のときは行番号200のサブルーチンへ飛びます。)

解説
〈式〉の値はGOTO文またはGOSUB文に続く〈行番号〉の列の何番目に移るかを示しています。たとえば〈式〉の値が2であれば、2番目の〈行番号〉の行に飛びます。式の値が0または指定した〈行番号〉の個数よりも多い場合は、次の実行可能な文を実行し、式の値が負の数または、255を超える場合はエラーとなります。

STOP
(ストップ)

➡ 基礎編82ページ

機能　プログラムの実行を中断します。

書式　STOP

解説
この文が実行されると、プログラムの実行は中断され、コマンド待ちの状態に戻ります。このとき、中断された行番号を、Break in 〈行番号〉 と表示します。
このときは、CONT命令により、プログラムの実行を再開することができます。
END文と違って、ファイルは閉じられません。

END
(エンド)

➡　基礎編29ページ

機能　**プログラムの実行を終わらせます。**

書式　**END**

解説

プログラムの実行を終わらせ、すべてのファイルを閉じた後、コマンド待ちに戻ります。END文はプログラム文中にいくつ書いてもかまいません。

プログラムの最後のEND文は省略することができますが、この場合、ファイルは閉じられません。

INPUT
(インプット)

➡　基礎編52ページ

機能　**キーボードから入力します。**

書式　**INPUT ［"〈プロンプト文〉";］〈変数名〉［,〈変数名〉…］**

文例　**INPUT "あなたの　なまえは" ;NA$**

(キーボードから文字変数NA$にあなたの名前を入力してください。)

解説

キーボードからの入力を読み、変数に代入します。INPUT文が実行されると、「？」疑問符を画面に表示して、キーボードからの入力待ちとなります。また〈プロンプト文〉がある場合は、疑問符の前にその文が表示されます。

データの入力は RETURN キーを押すことによって変数に入れられます。

LINE INPUT
(ライン インプット)

機能　**文字列を１行分、１つの文字変数に入力します。**

書式　LINE INPUT［"〈プロンプト文〉";］〈文字変数名〉

文例　LINE INPUT "なまえ、とし"; N$

(画面に「なまえ、とし」と表示した後、入力された文字列一行分をN$に入れます。)

解説

キーボードから入力された、全ての文字データを指定した文字変数に入力します。「一行」は画面上で２～３行にまたがってもかまいません。ただし文字数は255文字以内でなければなりません。入力は RETURN キーを押すことによって終了します。〈プロンプト文〉がある場合、入力を受けつける前に、その文を画面に出力します。INPUT 文と違って「？」は画面に出ません。

LINE INPUT 文実行中に CTRL ＋ C または CTRL ＋ STOP で実行を中断した後、CONT 命令で実行を再開させると、この文の最初より、プログラムを実行します。

サンプルプログラム

```
10  DIM A$(10)
20  FOR N=1 TO 10
30  LINE INPUT A$(N)
40  NEXT N: CLS
50  FOR N=1  TO  10
60  PRINT A$(N)
70  NEXT N
80  END
```

(10個の文字変数に値を入力すると、画面表示されます。)

PRINT
(プリント)

➡　基礎編17ページ

機能　**画面に式の値を表示します。**

書式　PRINT［〈式〉[{; または ,}] 〈式〉…］［{; または ,}］

文例　PRINT X*256+Y

(X*256+Yの値を画面に出力します。)

解説

文字式や数式の値を画面に表示します。数値を表示する場合、数値の最後にはスペースが１つつけ加えられ、先頭には符号用の１ケタ（正の数の場合スペースとなる）が加えられます。また数値は常に10進数で表示されます。省略形は「?」(疑問符)です。

PRINT命令の最後に「；」(セミコロン)または「，」(カンマ)をつけた場合、表示後の改行は行なわれません。また，PRINTとだけした場合、１行分のスペースを出力します。

PRINT USING
(プリント ユージング)

機能　フォーマット式に従って文字列や数値を表示する。

書式　PRINT USING "〈フォーマット式〉" ;〈式〉[; または , 〈式〉…]

解説

〈フォーマット式〉で用いられる書式制御用文字に従って、プリントされる文字や数値の出力書式を決定します。これらの書式制御用文字はそのまま文字として画面に出力されることはありません。

(1)　文字変数や文字列の書式制御

● ! (感嘆符)

与えられた文字変数や文字列の最初の一文字だけを出力します。

例：PRINT USING "! is the top."; "YAMA", "KAWA", "MORI"

● &(アンド)と␣(スペース)

必ずN個のスペース(0≦N)が2つの「&」にはさまれた形で用います。 このとき、 与えられた文字変数や文字列のうち先頭から(2＋N)個の文字を出力し、あまった文字は無視します。

また、指定した長さよりも文字変数や文字列の長さの方が短いときは領域内左づめで出力され、残った部分をスペースで満たします。

例：PRINT USING "&␣␣␣␣␣&"; "HITACHI", "MSX"

● @(アットマーク)

与えられた文字変数や文字列がそのまま出力されます。

例：PRINT USING "I LIKE @"; "CAT"

(2)　数値表示の書式制御

● #(シャープ)と .(ピリオド)

「#」は数字を表示する桁を、「.」は小数点を表示する位置を示します。したがって出力は固定小数点表示になります。「.」が無い場合は整数表示です。

指定した桁数よりも数値の桁数の方が短いときは領域内右づめで出力され、上位桁はスペースで満たされます。また小数部分で桁に足りない部分には0が出力されます。

長すぎる場合は、#の桁で四捨五入されます。

符号用として、1つの#が使用されることに注意してください。

例：PRINT USING "###.##␣"; 2.1, −2.5, 125.35

● ＋と−

数値書式制御文字の並びの前または後ろにつけます。ただし「−」は文字列の後ろにしかつけられません。

「＋」をつけたときは数値の前または後ろにその数値の符号が出力されます。「−」をつけたときは負数値の場合のみ負記号を数値の後ろに出力します。

「＋」と「−」は、それらを表示するための桁位置を必ず確保することに注意してください。

例：PRINT USING "+####.##"; 200, −25

● ,(カンマ)

小数点位置指定の「.」(ピリオド)の左側に置きます。これを指定した場合は、整数部分を右側から3桁ごとに「,」で区切ります。「,」を表示するたびにそのための桁位置を確保することに注意してください。

例：PRINT USING "＃＃＃＃＃＃＃＃＃，."；
3000000

● ＊＊(アスタリスク)
数値書式制御文字の並びの最初に置きます。このとき2桁分の桁位置を確保しますが、数値の上位桁に余白ができると、スペースの代わりにアスタリスク「＊」を出力します。

例：PRINT USING "＊＊＃＃＃＃，＃＃，."；
3750.25

● ￥￥(円記号)
「＊＊」と同様に使いますが、円記号「￥」は数値の直前に1つだけしか出力されません。浮動小数点形式の書式指定に「￥￥」を用いることはできません。負の数の場合、「－」は、￥のすぐ左横にあらわれます。

例：PRINT USING "￥￥＃＃＃＃＃＃＃＃，."；
10000

● ＊＊￥
「＊＊」、「￥￥」と同じように使います。3桁分の桁位置を余分に確保し，数値の上位に余白領域ができるとその1桁を「￥」で、残りを「＊」で満たします。

例：PRINT USING "＊＊￥＃＃＃＃．＃＃"；
3.14

● ＾＾＾＾(指数記号)
桁指定の「＃」の後ろに置きます。これによって浮動小数点形式の書式が指定され、「＃」によって指定するのは仮数部の表示桁となります。

例：PRINT USING "＃＃．＃＃＃＃＾＾＾＾"；
－12.345

注）表示しようとする数値の桁数が、指定した表示領域を超えてしまった場合は、数値の前に「％」が出力されます。これは四捨五入による丸めが原因となった場合についても同様です。
〈フォートマット式〉の中に、上に述べた書式制御用文字以外の文字が現われる場合、それは制御用文字定数とみなしその文字がそのまま出力されます。

LPRINT
(エル プリント)

➡ 基礎編37ページ

機能　プリンタに式の値を出力します。

書式　LPRINT [〈式〉 [；または ， 〈式〉…] [；または ，]]

文例　LPRINT A, B；C$；D$
(変数A、B、C$、D$の値をプリンタに出力します。「，」「；」の違いはPRINT文と同じ意味です。)

解説
定数、変数や式の値をプリンタに出力します。
〈式〉がない場合は改行のみを行ないます。
PRINT文と同じように、区切り記号として、「；」、「，」や「␣」(空白)を使うことができます。
LPRINT文の最後が「；」で終っていても、STOP文や、CTRL ＋ STOP でプログラムの実行を中断した場合には改行されます。

注）省略形として、L？は使えません。

LPRINT USING
(エル プリント ユージング)

機能　〈フォーマット式〉に従って、式の値をプリンタに出力します。

書式　LPRINT USING "〈フォーマット式〉";〈式〉[; または , 〈式〉…]

文例　LPRINT USING "＃＃＃.＃＃";12.5;1.0E+2;2.3E−2

解説

プリンタに出力をするという点を除いては、PRINT USING命令と同じです。PRINT USING命令を参照してください。

READ
(リード)

➡ 基礎編129ページ

機能　DATA文中に書かれたデータを読み込み、変数に割り当てます。

書式　READ 〈変数名〉[, 〈変数名〉…]

文例　READ A$, B$, C
(DATA文で定義されたA$、B$、Cの値を読み込みそれぞれの変数に代入します。)

解説

DATA文中に書かれたデータをその順に、READ文中の変数に割り当てます。代入すべき変数が数値変数である場合、対応するデータも数値定数でなければなりません。もし、READ文の変数の数が、一連のDATA文のデータの数を超えてしまった場合は、Out of DATAエラーが生じます。DATA文のデータが余った場合には、残りは無視されます。RESTORE文により、DATA文のデータを読み直すことができます。

DATA
(データ)

➡ 基礎編129ページ

機能 READで使用するデータを定義します。

書式 DATA <定数> [，<定数>…]

文例 DATA 1453，622

(数値1453、622をデータとして定義します。)

解説

DATA文はREAD文で読み込まれる<定数>を用意するための文です。
DATA文は非実行文で、プログラム中に自由に置くことができます。またいくつあってもかまいません。1つのDATA文には、カンマで区切って1行に入るだけ（255文字以内）の<定数>を書くことができます。READ文は行番号の小さいDATA文から順に<定数>の読み込みを行ないます。

注）DATA文の中の文字列に「，」(カンマ)、「.」(ピリオド）や文字列の先頭にスペースを含む場合には、それぞれの文字列全体を「"」(ダブルクォート)で囲まなければなりません。

RESTORE
(リストア)

➡ 基礎編131ページ

機能 READ文で読み込まれるデータの開始行を指定します。

書式 RESTORE [<行番号>]

文例 RESTORE 120

(行番号120のDATA文から、データを読み込みます。)

解説

同じ一群のデータを複数回にわたり読み込む場合などに使用します。RESTORE文が実行されると、次からのREAD文は<行番号>以降にある最初の、DATA文から読み込みを行ないます。<行番号>を省略した場合、プログラム中最初のDATA文から読み込みを行ないます。

DIM
(ディメンジョン)

➡ 基礎編123ページ

機能　**配列の要素の大きさを指定し、メモリ領域を割り合てます。**

書式　**DIM 〈変数名〉(〈添字の最大値〉[，〈添字の最大値〉]…)**

文例　**DIM　A(30)，B$(5)**

(変数Aに31、B$に6の配列を定義します。)

解説

配列の次元数と添字の最大値を設定します。同時にメモリ上にその配列の領域を割り当てます。配列の次元数はそれぞれの変数の〈添字の最大値〉の数によって決定されます。添字の最小値は常に0ですから、1つの配列での要素の数は各次元の、〈添字の最大値〉に1を加えたものの積となります。

DIM文で宣言されない配列変数が使われた場合、添字の最大値は10に設定されます。

ERASE
(イレース)

機能　**プログラム中の配列を消去します。**

書式　**ERASE 〈配列名〉[，〈配列名〉…]**

文例　**ERASE X，Y，Z**

(配列変数X、Y、Zを消去します。)

解説

〈配列名〉で指定した配列をメモリから消去します。この命令によって、以前配列に使われていた領域を、別の用途に使うことができます。

また、一度ERASEにより消去された配列をDIM文により再び配列宣言することも可能です。

(ERASEされていない配列を2度宣言するとRedimensioned array　エラーになります。)

サンプルプログラム

```
10  DIM X(20)
20  FOR N =1 TO 20
30  X(N)=N-2
40  NEXT N
50  ERASE X
60  DIM X(30)
70  FOR N=1 TO 30
80  X(N)=N*3
90  NEXT N
100 END
```

(変数Xの配列を20と宣言していましたが、行番号50で、配列を消去し、行番号60で改めて、配列を30と宣言します。)

DEF FN
(デファイン ファンクション)

機能 ユーザ関数の定義をします。

書式 DEF FN <名前> [(<仮引数> [, <仮引数>…])]=<関数の定義式>

文例 DEF FNA (X , Y)=SQR (X*X+Y*Y)
(X*X+Y*Yの平方根を返すFNA (X , Y) という関数を定義します。)

解説

ユーザが定義する式を、FN<名前>という関数として宣言します。その結果他の関数と同様に取り扱うことができます。
仮引数とは、その関数が呼ばれたときに、実際の値と置き換えられる変数です。ここに書かれた変数名と同じ名前の変数名が、他の場所で使われていても、この文は影響を受けません。
<名前>は変数名と同様の条件を満たさなければなりません。また定義式と名前の型は一致していなければなりません。
DEF FN文は、定義される関数が使用される前に実行されなければなりません。

注) DEF FN文にエラーがあると、エラーメッセージは関数を引用している行を指す場合がありますので注意してください。

DEFINT/SNG/DBL/STR
(デファイン インテジャー/シングル/ダブル/ストリング)

機能 変数名の型宣言を行ないます。

書式 DEF <型><変数名の頭文字の範囲> [, <変数名の頭文字の範囲>…]

文例 DEFINT I, J
(I, Jで始まる変数名が整数型であることを宣言します。)

DEFSNG S, T, V
(変数名がS、T、Vで始まる変数が単精度型であることを宣言します。)

DEFDBL D−G
(変数名がDからGまでの頭文字で始まる変数が倍精度型であることを宣言します。)

DEFSTR A, E−F
(変数名がAおよびEからFまでの頭文字で始まる変数が文字型であることを宣言します。)

解説

指定した範囲の頭文字で始まる変数名および配列名の型を宣言します。ただし、変数名に属性文字の「%」整数型、「#」単精度型、「！」倍精度型、「＄」文字型をつけた場合、変数の型は型宣言よりも優先します。

〈型〉は

INT ……**整数型**
SNG ……**単精度型**
DBL ……**倍精度型**
STR ……**文字型**

のいずれかを指定します。

型の宣言を行なわなかった文字で始まり属性文字を持たない変数名の変数はすべて倍精度型とみなされます。

注）DEFと〈型〉の間は、スペース(空白)をあけると、Syntax errorが出ます。また「－」（マイナス）でつないだ頭文字はアルファベット順でないとSyntax errorが表示されます。

サンプルプログラム

```
10  DEFINT A : DEFSNG B
20  DEFDBL C : DEFSTR D
30  A=6 : B=3.1415927 : C=9
     : D="SAKAI"
40  PRINT A ; B ; C
50  PRINT B／A ; A／C , D
60  END
```

(変数Aを整数型、Bを単精度型、Cを倍精度型、Dを文字型に宣言して、行番号30以下を実行します。)

MID$
(ミドル ドル)

機能　**文字変数の1部を別の文字列で置き換えます。**

書式　MID$ (〈**文字変数**〉, 〈**引数 1**〉　[, 〈**引数 2**〉]) = 〈**文字式**〉

文例　MID$ (A$, 2,4) = "WHAT"
(文字変数A$の左から2文字目からの4文字を "WHAT" に置き換えます。)

解説

〈文字変数〉の〈引数1〉番目から〈引数2〉個の文字列を、〈文字式〉と交換します。文字は左側から数えることに注意してください。

〈引数1〉は、〈文字変数〉の長さ以下の数を指定しなければなりません。〈引数2〉は0～255の範囲の数が指定できます。

省略すると、〈引数1〉番目から右を全て与えます。

サンプルプログラム

```
10  A$="I LIKE A DOG"
20  PRINT A$
30  MID$ (A$, 10, 3)="CAT"
40  PRINT A$
50  END
```

(A$の文字列の中のDOGをCATに置き換えて表示します。)

SWAP
(スワップ)

機能　2つの変数の値を交換します。

書式　SWAP 〈変数名〉,〈変数名〉

文例　SWAP A, B
(変数Aの値と、変数Bの値を交換します。)

解説

SWAPの後に書かれた2つの変数の値を交換します。
変数の型は、一致していなければなりません。
配列変数も使用することができます。

サンプルプログラム

```
10  A=10:B=3
20  SWAP A,B
30  PRINT "A= ";A
40  PRINT "B= ";B
50  END
```

(変数Aの値と、変数Bの値を交換して画面に表示します。)

入出力ステートメント

MAXFILES
(マックス ファイルズ)

機能　同時に開くファイルの最大値を指定します。

書式　MAXFILES＝〈式〉

文例　MAXFILES＝15
(同時に開くファイルの数を15に指定します。)

解説
〈式〉の値は、0から15の範囲です。MAXFILES＝0と指定すると、SAVE,LOADのみが有効となります。
この命令を実行しないときには、開くことのできるファイル数は1です。

OPEN
(オープン)

機能　**プログラムで使用できるようにファイルを開きます。**

書式　**OPEN ”〈デバイスディスクリプタ〉[〈ファイル名〉]” [FOR〈モード〉] AS [#]〈ファイル番号〉**

文例　**OPEN ”CAS：SAMPLE” FOR OUTPUT AS#1**

(ファイル番号1のファイルを出力モードで開いて、カセットレコーダのファイル名 ”SAMPLE” をプログラムで使用可能にします。)

解説

〈デバイスディスクリプタ〉で指定したファイルを、指定した〈ファイル番号〉で開き、ベーシックプログラム中で使える状態にします。

〈デバイスディスクリプタ〉には以下の5種類が使用できます。「：」も含まれますので注意してください。

CAS：	カセットレコーダ
CRT：	画面
GRP：	グラフィックスクリーン
LPT：	ラインプリンタ
MEM：	RAMディスク

〈モード〉は以下の3種類があります。

OUTPUT	出力モード
INPUT	入力モード
APPEND	追加モード

〈ファイル番号〉は1～15の任意の整数を使用できます。ただし、2つ以上を指定するときにはMAXFILES命令で先に宣言しておく必要があります。

CLOSE
(クローズ)

機能　**入出力ファイルを閉じます。**

書式　**CLOSE [[#]〈ファイル番号〉[,〈ファイル番号〉…]]**

文例　**CLOSE#1, 2, 5**

(ファイル番号1、2、5のファイルを閉じます。)

解説

指定した番号の入出力ファイルを閉じ、その番号のバッファを他のファイルで使用できるようにします。〈ファイル番号〉を指定しないでCLOSE文を実行すると、開かれているファイルを全て閉じます。

PRINT♯
(プリント シャープ)

機能　指定したファイルに式の値を出力します。

書式　PRINT♯〈ファイル番号〉,[〈式〉[; または , 〈式〉…][; または ,]]

文例　① PRINT♯1,"X=";X
（番号1のファイルに、"X="の文字列と、変数Xの値を出力します。）

② PRINT♯1, X, Y, Z;
（番号1のファイルに、変数X、Y、Zの値を出力します。）

解説

〈ファイル番号〉で指定したファイルに〈式〉の値を出力します。ファイルはOPEN命令で出力モードに開かれていなければなりません。
ファイルに出力する点を除いてはPRINT命令と同じです。

サンプルプログラム

```
10  OPEN "CAS : SAMPLE" FOR OUTPUT
    AS #1
20  A$="TSUZUKIAOYAMAMAEKAWA"
30  PRINT #1, A$
40  CLOSE
50  END
60  MAXFILES=2
70  OPEN "CAS : SAMPLE" FOR INPUT
    AS #1
80  B$=INPUT$ (7, #1)
90  OPEN "CRT : " FOR OUTPUT AS #2
100 PRINT #2, B$
110 CLOSE
120 END
```

（行番号10～50：カセットレコーダに、行番号20のデータを記録します。行番号60～120：カセットレコーダに記録したデータを読み込み、その内容を画面に表示します。）

PRINT♯ USING
(プリント シャープ ユージング)

機能　指定したファイルに、フォーマットに従って式の値を出力します。

書式　PRINT ♯〈ファイル番号〉, USING "〈フォーマット式〉"[;〈式〉[; または , 〈式〉…][; または ,]]

文例　①PRINT #1,USING "###.##";X,Y,Z
（番号1のファイルに、変数X，Y，Zの値を固定小数点形式で出力します。）
②PRINT #2,USING "##.###^^^^";2^20
（番号2のファイルに、2の20乗の値を浮動小数点形式で出力します。）

解説
<ファイル番号>で指定したファイルに<フォーマット式>に従って<式>の値を出力します。

詳しい説明は、PRINT USING命令、PRINT #命令を参照してください。

INPUT
（インプット シャープ）

機能　**入力ファイルからデータを読み込みます。**

書式　INPUT# <ファイル番号>, <変数名> [, <変数名>…]

文例　INPUT#5,A,B
（ファイル番号5のファイルから、データを読み込み変数A、Bに代入します。）

解説
指定したファイルからデータを入力します。指定するファイルは、あらかじめ開かれていなければなりません。

サンプルプログラム
```
10  OPEN "CAS : SAMPLE" FOR OUTPUT
    AS #1
20  PRINT #1,1;2;3;4;;5;6;7;8;9;0
30  CLOSE
40  END
50  MAXFILES=2
60  OPEN "CAS : SAMPLE" FOR INPUT
    AS #1
70  OPEN "CRT : " FOR OUTPUT AS #2
80  INPUT #1, A, B, C, D
90  PRINT #2, A;B;C;D
100 CLOSE
110 END
```
（行番号10～40：カセットレコーダに行番号20の数値データを記録します。行番号50～110：記録された数値データを、変数A、B、C、Dに代入し内容を画面に表示します。）

LINE INPUT♯
(ライン インプット シャープ)

機能　入力ファイルから１行分のデータを読み込みます。

書式　LINE INPUT♯〈ファイル番号〉,〈文字変数名〉

文例　LINE INPUT♯5, A$
(ファイル番号５のファイルから、１行分のデータを読み込み、変数A$に代入します。)

解説

〈ファイル番号〉で指定したファイルから１行読み込み、変数に代入します。ここでの１行とは改行コードを読み込むまでの入力全てです。ただし、文字変数には255までしか入力できません。
ただし、この場合、前もってCLEAR文で十分、大きく文字領域を確保する必要があります。
改行コードとは、キャラクタコード13(&H0D)の文字、またはキャラクタコードが13(&H0D)の文字と10(&H0A)の文字が連続したものです。
指定したファイルは〈ファイル番号〉に対して、あらかじめ入力モードで開かれていなければなりません。

サンプルプログラム

```
10  OPEN "CAS : SAMPLE" FOR OUTPUT
    AS #1
20  A$= "TSUZUKIAOYAMA"
30  PRINT #1, A$
40  CLOSE
50  END
60  MAXFILES=2
70  OPEN "CAS : SAMPIE" FOR INPUT
    AS #1
80  OPEN "CRT : " FOR OUTPUT AS #2
90  LINE INPUT #1, A$
100 PRINT #2, A$
110 CLOSE
120 END
```

(行番号10〜50：カセットレコーダへ文字列データA$を記録します。行番号60〜120：カセットレコーダから、行番号50までで記録したデータを１行分読み込み変数A$に代入してその内容を画面に表示します。)

INPUT$
（インプット ドル）

機能　指定した文字数を、指定したファイルから入力します。

書式　INPUT$（〈文字数〉,［#］〈ファイル番号〉）

文例　INPUT$（5，#1）

解説

〈文字数〉で指定した数の文字を、〈ファイル番号〉で指定したファイルから読み取ります。指定したファイルは、開かれていなければなりません。

サンプルプログラム

```
10  OPEN"CAS：SAMPLE" FOR OUTPUT
    AS #1
20  A$="67890123456789012
    34567890123456789000
    00"
30  PRINT #1，A$
40  CLOSE
50  END
60  MAXFILES=2
70  OPEN"CAS：SAMPLE"FOR INPUT
    AS #1
80  B$=INPUT$(15，#1)
90  OPEN"CRT："FOR OUTPUT AS #2
100  PRINT #2,B$
110  CLOSE
120  END
```

（行番号10～50：カセットレコーダに、行番号20のデータを記録します。行番号60～120：カセットレコーダから、行番号50までのプログラムで記録したデータのうち15文字分を入力し、画面に表示します。）

画面・グラフィックステートメント

PSET命令、PRESET命令、LINE命令、CIRCLE命令、PAINT命令の座標（画面位置）には、相対的位置を表わすSTEPも使えます。186ページ、105ページのPUT SPRITE命令をご覧ください。

CLS
（クリア スクリーン）

➡ 基礎編24ページ

機能　画面をクリアし、カーソルをホーム位置へ移動させます。

書式　CLS

解説

全ての画面モードでその表示内容を消去し、カーソルをホーム位置（画面左上隅）に置きます。

LOCATE
(ロケイト)

➡ 基礎編47ページ

機能　カーソルの位置および働きを指定します。

書式　LOCATE 〈水平位置〉, 〈垂直位置〉 [, 〈カーソルモード〉]

文例　LOCATE　15, 12
(カーソルを、水平位置15、垂直位置12の場所に移動します。)

解説

カーソルを指定した位置へ移動します。位置はテキストモードのときの画面の位置です。40×24テキストモードの場合、〈水平位置〉の範囲は0～39、〈垂直位置〉の範囲は0～22(KEY　OFFで23)です。80×24テキストモードの場合、〈水平位置〉の範囲は0～79、〈垂直位置〉の範囲は0～22(KEY　OFFで23)です。32×24テキストモードの場合、〈水平位置〉の範囲は0～31、〈垂直位置〉の範囲は0～22(KEY　OFFで23) です。また指定された位置が画面の範囲を超えていると縦横とも、それを超えない最大の値に設定し直されます。

なお原点（0, 0）は左上隅です。〈カーソルモード〉で0を指定するとカーソルを表示しません。1を指定するとカーソルを表示します。省略すると、1が設定されます。

WIDTH
(ウィドス)

機能　1行に表示する文字を設定します。

書式　WIDTH 〈表示文字数〉

文例　WIDTH　25
(画面表示文字数を1行25文字にします。)

解説

画面1行に表示する文字数を〈表示文字数〉に設定します。(SCREEN 0 の40×24テキストモードで1～40、SCREEN 0 の80×24テキストモードで41～80、SCREEN 1の32×24テキストモードでは、1～32までの値が指定できます。)

注) 各モードで文字数を最大にすると、テレビによっては文字が画面からはみ出ることがあります。

COLOR
(カラー)

➡基礎編85ページ

機能　**画面の色を指定します。**

書式1　COLOR ［〈**文字色コード**〉］［ ,〈**背景色コード**〉］［ ,〈**境界色コード**〉］

書式2　COLOR＝(**パレット番号**,**Rコード**,**Gコード**,**Bコード**)

書式3　COLOR＝RESTORE

書式4　COLOR ［＝NEW］

文例1　COLOR　15, 1, 8
(文字色を白、背景色を黒、境界色を赤に設定します。)

文例2　COLOR＝(4, 7, 1, 1)
(カラーパレット4に赤色を割り当てます)

解説1

画面全体の文字の色と背景全体の色を指定します。〈文字色コード〉〈背景色コード〉〈境界色コード〉とも以下の0～15（SCREEN 6では0～3までの値で指定します。

カラーコード

0	透明（境界色と同じ）	8	赤
1	黒	9	明るい赤
2	緑	10	暗い黄
3	明るい緑	11	明るい黄
4	暗い青	12	暗い緑
5	明るい青	13	紫
6	暗い赤	14	灰
7	水色	15	白

解説2

SCREEN 6では0～3、SCREEN 8以外では0～15のパレットに色を割り当てることができます。RGBのそれぞれのコードは、0～7の値(RGB各3ビット)で指定し組み合わせにより512色の中から選んで自由に設定することができます。
なおSCREEN 8では0～255の値で色を指定します。(SCREEN 8ではカラーパレットは使用できません。)

　　0 0 0 0 0 0 0 0　8ビットの色コード
　　G G G R R R B B

例えば赤は&B 00111101（＝61）となります。

解説3

画面データを（BLOAD）で読み込んだ時にパレット情報が含まれていた場合には

　　COLOR＝RESTORE

によってパレット設定をセーブした時の状態にすることができます。パレットを電源投入時の状態に戻すには

　　COLOR（＝NEW)

を実行してください。(　) 内は省略できます。

SCREEN
(スクリーン)

➡基礎編90ページ

機能　画面モード、スプライトの大きさ、キーのクリック音の有無、カセットレコーダーへのボーレイト、専用プリンタの有無を指定します。

書式　SCREEN [〈画面モード〉] [, 〈スプライトサイズ〉] [, 〈キークリック〉] [, 〈ボーレイト〉] [, 〈プリンタオプション〉]

文例　SCREEN 2, 3, 1, 1, 1

(画面モードをハイリゾリューションモードに設定し、スプライトサイズ16ドット×16ドットの拡大表示、キーを押したときにクリック音を発生させます。ボーレイトは1200ボーモード、使用プリンタがMSX専用以外のものであることを指定します。)

解説

① 〈画面モード〉は0～8の数字が指定できます。

0：39文字×24行　(最大80文字×24行)
　テキストモード
1：29文字×24行　(最大32文字×24行)
　テキストモード
2：256ドット×192ドット
　ハイリゾリューションモード
3：64ブロック×48ブロック
　ローリゾリューションモード
4：256ドット×192ドット
　ハイリゾリューションモード
5：256ドット×212ドット
　ビットマップグラフィックモード
6：512ドット×212ドット
　ビットマップグラフィックモード
7：512ドット×212ドット
　ビットマップグラフィックモード
8：256ドット×212ドット
　ビットマップグラフィックモード

・テキストモード（画面モード0，1）ではグラフィック関係の命令を実行するとエラーになります。必ずSCREEN命令を先に実行してください。画面モード1ではスプライトを表示することができます。

・INPUT文を実行したいときおよびコマンド待ちの状態になったときにはテキストモードに変更されます。省略すると1とみなします。

・画面モード4は、スプライト機能が強化されている(横方向に9個まで表示可)ハイリゾリューションモードです。

・各画面モードで使うことのできる色は次のようになります。

0：16パレット中の2色
1：16パレット中の3色

(VPOKE命令などを使うと16色も可能)

2～5，7：16パレットのすべて
6：0～3パレット
8：256色固定

② 〈スプライトサイズ〉真は０～３の数字が指定できます。

0：８ドット×８ドット標準
1：８ドット×８ドット拡大
2：16ドット×16ドット標準
3：16ドット×16ドット拡大

・この命令を実行するとSPRITE$で設定したスプライトパターンは消えてしまいます。
省略すると、電源投入時では０をまたは以前に指定したサイズとみなします。
画面のモードを変えずにスプライトの大きさだけを設定するには以下のように指定します。

```
SCREEN , 2
```

③ 〈キークリック〉はキーを押したときのポッポッというクリック音発生の有無を指定します。

0, 1が指定できます。

0：クリック音を発生しない
1：クリック音を発生させる

省略すると、電源投入時では１を、または以前に指定した値をみなします。

④ 〈ボーレイト〉はカセットレコーダへの書き込み速度を指定します。
1, 2の２種類があります。

1：1200ボー
2：2400ボー

省略すると、電源投入時では１をまたは以前に指定したボーレイトとみなします。

・カセットレコーダから読み込む場合、ボーレイトは自動的に決定されますので書き込んだ時のボーレイトの値を記録しておく必要はありません。

⑤ 〈プリンタオプション〉は使用するプリンタが本機専用のものか、そうでないかを指定します。０以外の数字を指定するとプリンタはMSX専用ではないとみなされ、グラフィックキャラクターはスペースに、ひらがなはカタカナに換えて、プリンタに出力されます。
省略すると、電源投入時では０を、または、以前に指定した値とみなします。

・画面モード、スプライトサイズを変えずにキークリック、ボーレイト、プリンタオプションを設定するには、以下のようになります。

```
SCREEN , , 1 , 2 , 0
```

(キーを押した時クリック音を発生させ、ボーレイトは2400ボーモード、使用プリンタはMSX専用のものであることを設定します。)

PSET
(ピー セット)

➡基礎編92ページ

機能　**指定した位置に点を打ちます。**

書式　PSET［STEP］(**水平位置，垂直位置**)［，〈**カラーコード**〉］［，〈**論理演算子**〉］

文例　PSET (50，50)，6

(グラフィック画面上，50，50の位置に暗い赤で点を打ちます。)

解説

画面上に〈カラーコード〉で指定した色で点を打ちます。

SCREEN　2，4：〈水平位置〉0～255，
　　　　　　　〈垂直位置〉0～191
SCREEN　5，8：〈水平位置〉0～255，
　　　　　　　〈垂直位置〉0～211
SCREEN　6，7：〈水平位置〉0～511，
　　　　　　　〈垂直位置〉0～211
SCREEN　3，　：〈水平位置〉0～255，
　　　　　　　〈垂直位置〉0～191

の範囲で指定した点が含まれるブロックが表示されます。

〈カラーコード〉はそれぞれのスクリーンモードでの指定方法に準じます。(COLOR参照)

省略した場合、電源投入時に決められている文字色、またはその後COLOR命令で指定された文字色が選択されます。

STEP，〈論理演算子〉についてはLINEを参照してください。

注) テレビ信号の特性により指定した色と異なることがあります。

PRESET
(ピー リセット)

➡基礎編94ページ

機能　**指定した位置の点を消します。**

書式　PRESET［STEP］(**水平位置，垂直位置**)［，〈**カラーコード**〉］［，〈**論理演算子**〉］

文例　PRESET (100，20)

(グラフィック画面上、100，20の位置に打たれている点を消します。)

解説

〈カラーコード〉を省略した場合、指定した画面上の位置にある点を消します。

〈カラーコード〉をつけたときはPSET命令と同じ機能を持つことになります。STEP，〈論理演算子〉については、LINEを参照してください。

LINE
（ライン）

➡基礎編95ページ

機能　指定した2点間に直線を引きます。またその2点を対角とする長方形を描いたり、色で塗りつぶしたりします。

書式　LINE［STEP］（始点位置）－［STEP］（終点位置）［，〈カラーコード〉］［，〈オプション〉］［，〈論理演算子〉］

文例　① LINE（20，30）－（100，100）
（座標20，30から100，100まで文字色で線を引きます。）

② LINE（10，10）－（100，100），1，B
（座標10，10と100，100を対角とする長方形を、黒で描きます。）

③ 10　LINE（10，10）－（100，100）
　 20　LINE－STEP（50，－30）
（座標10，10から100，100まで文字色で線を引き、終点位置100，100を原点としてそこから50，－30の位置まで線を引きます。）

解説

〈始点位置〉から〈終点位置〉まで〈カラーコード〉で指定した色で直線を引きます。〈オプション〉にBを指定すると〈始点位置〉と〈終点位置〉を対角とする長方形を描き、BFを指定すると長方形の中を指定したカラーコードの色で塗りつぶします。〈カラーコード〉が省略された場合は、電源投入時に決められている文字色、またはその後COLOR命令で指定された文字色が選択されます。

(〈始点位置〉)が省略された場合は、以前に指定した(〈終点位置〉)が(〈始点位置〉)とみなされます。ただしSTEP命令を使った場合には、以前に指定した(〈終点位置〉)が原点（0，0）として設定されます。電源投入時(〈始点位置〉)は（0，0）に設定されます。

LINEの最後に〈論理演算子〉を指定すると色をビット単位の信号として以下のように操作することができます。ただしSCREEN 5～8に限られます。

(COLOR参照)

Aを指定色、Bを指定色の下になる色とした時

・AND：A　AND　B
　例　SCREEN 5の場合
　明るい水色(0,7,7) AND 黄色(7,7,0)＝緑色(0,7,0)
〔000, 111, 111〕AND〔111, 111, 000〕＝〔000, 111 000〕

・OR：A　OR　B
　例　SCREEN 8の場合
　赤色（28）OR 明るい緑（224）＝黄色（252）
〔00011100〕OR〔11100000〕＝〔11111100〕

・XOR：NOT　A　AND　B
　例　SCREEN 5の場合
　暗い青 XOR 緑色＝暗い赤

・PSET：A（指定色がそのまま表示されます。）

・PRESET：NOT　A
　例　SCREEN 5の場合
　灰色 PRESET＝黒

CIRCLE
（サークル）

➡基礎編97ページ

機能　円、だ円、弧を描きます。

書式　CIRCLE ［STEP］（〈中心位置〉）〈半径〉［〈，カラーコード〉］［，〈開始角度〉］［，〈終了角度〉］［，〈扁平率〉］

文例　CIRCLE （120，100），30，1，0，6.28，0.5
（座標120、100を中心に、横方向の半径30で、横長のだ円を描きます。）

解説

〈中心位置〉を中心に、〈半径〉で指定される大きさの円を描きます。

〈カラーコード〉が省略された場合には、COLOR命令で指定された文字色となります。

〈開始角度〉〈終了角度〉はラジアンで指定します。
また、〈開始角度〉〈終了角度〉が負の数値で指定されると〈開始角度〉と中心と〈終了角度〉を結ぶ扇形を描きます。
ただし〈開始角度〉，〈終了角度〉を－0と指定しても中心とは結ばれません。
だ円を描く場合の〈扁平率〉は1より小さい場合は横長のだ円、1より大きい場合は縦長のだ円となり〈半径〉は、〈扁平率〉が1より小さい場合は横方向の長さ、1より大きい場合は縦方向の長さとなり、省略すると1とみなします。
STEP命令を使うと以前にLINE、PSETなどで指定したいちばん最後の位置を原点（0，0）と設定します。
〈開始角度〉は省略すると0とみなします。
〈終了角度〉は省略すると6.28とみなします。

〈開始角度〉　〈終了角度〉

1.57（π／2）

3.14（π）

0または 6.28（2π）

4.71（3／2π）

注）SCREEN 6，7で横512ドット×縦212ドットの場合には円をまるく見せるために扁平率は標準で0.5に設定されています。

PAINT
（ペイント）

➡基礎編99ページ

機能　指定された境界の中を塗りつぶします。

書式　PAINT［STEP］〈開始位置〉［，〈カラーコード１〉］［，〈カラーコード２〉］

文例　① PAINT（100，51），8，1

（黒で囲まれた図形の中を赤で塗りつぶします。座標100，51が図形の外にある場合は図形の外側を赤で塗りつぶします。（SCREEN３，５～８）

② PAINT（50，80），3

（明るい緑の図形を同じ色で塗りつぶします。この場合、〈カラーコード２〉は省略できます。）

（SCREEN2,4）

解説

〈カラーコード２〉の色で囲まれた図形の中または外を〈カラーコード１〉の色で塗りつぶします。開始位置が図形の内側にある場合には図形の中を図形の外側にある場合には、図形の外側を塗りつぶします。

SCREEN２、４では〈カラーコード１〉と〈カラーコード２〉は同じでなければなりません。

この場合〈カラーコード２〉は省略できます。〈カラーコード１〉は省略すると、文字色と同じとみなします。

SCREEN３，５～８では〈カラーコード１〉と〈カラーコード２〉は別々に指定できます。

〈カラーコード１〉を省略すると文字色と同じとみなします。〈カラーコード２〉を省略すると〈カラーコード１〉と同じとみなします。

PAINTを実行したときに指定した〈カラーコード２〉で囲まれた図形（SCREEN２、４の場合には〈カラーコード１〉）がないと画面一面が指定した色で塗りつぶします。

STEP命令を使うと以前にLINE、PSETなどで指定したいちばん最後の位置を原点（0，0）と設定します。

PUT SPRITE
（プット スプライト）

➡基礎編105ページ

機能 指定したスプライトを画面に表示します。

書式 PUT SPRITE〈スプライト面番号〉［，〈STEP〉〈表示位置〉］［，〈カラーコード〉］［，〈パターン番号〉］

文例 PUT SPRITE　0，（100，100），8，1
（パターン番号1をスプライト面0の（100、100）の画面位置に赤で表示します。）

解説

〈パターン番号〉で指定したスプライトを〈スプライト面番号〉で指定したスプライト面の〈表示位置〉に〈カラーコード〉の色で表示します。
〈スプライト面番号〉は0から31の値がとれます。
〈表示位置〉には次の2通りの書き方があります。

1）STEP（〈X方向の増分〉，〈Y方向の増分〉）現在の表示位置から相対的にスプライトを移動させたい場合に用います。
2）（〈X位置〉，〈Y位置〉）
スプライトを画面を絶対位置に表示させる場合に用います。

〈X位置〉は－32～255、〈Y位置〉は、－32～191の範囲で使用します。〈Y位置〉に208（SCREEN 4～8では216）が指定された場合、指定されたスプライト面の背後にあるすべてのスプライトは画面に表示されなくなります。これは208（216）以外の値が、そのスプライトに設定されるまで続きます。
また、〈Y位置〉209（SCREEN 4～8では217）を指定した場合、指定されたスプライトが画面に表示されなくなります。
1つのスプライト面には、1つのパターンだけしか表示できません。すでにパターン表示してある面を指定すると、先に表示されていたパターンは消されます。
パターンはスプライト面の0～31に同時に表示できますが横方向に5個以上（SCREEN 4～8では9個以上）のパターンが並ぶと5個目（9個目）以降のパターンは画面から消えます。
〈表示位置〉を省略した場合、以前に指定した同じスプライト面の位置とみなします。指定していなければ画面には表示されなくなります。〈カラーコード〉はCOLOR命令で使われるものと同じです。省略すると文字色と同じになります。SCREEN 4～8での各スプライト面では1ライン毎に色を付けることができます。（COLOR　SPRITE参照）
〈パターン番号〉はSPRITE$命令で作成したスプライトの番号を使用します。スプライトサイズ（SCREEN命令で設定します）が0か1の場合は0～255まで、2，3の場合は0～63までが使用できます。
また、〈パターン番号〉を省略した場合は〈スプライト面番号〉と同じとみなされます。

COLOR SPRITE
（カラー スプライト）

機能　スプライトの色を指定する。

書式　COLOR　SPRITE［$］（スプライト面番号>）［＝（式）］

文例1　COLOR　SPRITE$（0）＝CHR$（1）＋CHR$（6）

（スプライト面の0のスプライトの1ライン目を黒に2ライン目を赤に指定し他は以前の色のままです。）

文例2　COLOR　SPRITE（1）＝6

（スプライト画面1を赤に指定します。）

解説

SCREEN 4～8の場合のみスプライトの色指定することができます。カラーコードについては、COLORを参照してください。

文例1のように「$」をつけると、一ラインごとに色指定することもできます。

SCREEN 8の場合には、以下のようなカラーパレットを使用します。

0：	黒	8：	肌色
1：	暗い青	9：	青
2：	暗い赤	10：	赤
3：	暗い紫	11：	紫
4：	暗い緑	12：	緑
5：	暗い水色	13：	水色
6：	暗い黄色	14：	黄
7：	灰色	15：	白

カラーコードが15を超える場合上位4ビットは次のような意味を持ちます。

bit 7（&B1000XXXX）：32ドット左シフト表示

bit 6（&B0100XXXX）：PUT　SPRITEを実行したとき連続したスプライト画面を同時に動かします。このとき重なった部分の色コードはORされて表示され衝突は検出しません。

スプライト画面0（水色）

a ORされて白になります。

b スプライト画面1（赤）

スプライトが重なった場合上図のように表示されます。

全てのSPRITE$をSTRING$（32,255）として、スプライト画面0が&B00000111（水色）、スプライト画面1が&B01001000（bit 6がONの赤）の場合、aの重なった場所はOR化されて白（&B00001111）になります。

bの部分は同じ水平ライン上にbit 6がOFFでスプライト画面番号が若いスプライトがないために表示されなくなります。

またbit 6がONのスプライトが連続していると1つのPUT　SPRITE文でそれらを動かすことができます。

たとえばスプライト画面0のbit 6がOFF、その後1～4までがONだとすると…

PUT　SPRITE 0,（X，Y）…とするだけで0～4までのスプライトを同時に動かすことができます。

bit 5（&B0010XXXX）衝突を検出しない。

bit 4 未使用

スプライト画面全体の色を変えるには文例2のようにスプライト画面番号とカラーを指定してください。

PUT KANJI
(プット カンジ)

➡基礎編132ページ

機能　SCREEN 5～8において漢字を表示します。

書式　PUT KANJI［(〈表示位置〉)］，〈漢字コード〉［，〈カラーコード〉］［，〈論理演算子〉］［，〈モード〉］

解説

漢字コードは、JIS漢字コードで&H2121から&H7E7Eまでの値を使います。

〈モード〉は以下のような意味を持ちます。

0：16×16ノーマル
1：8×16偶数ライン
2：8×16奇数ライン

漢字ROMカートリッジがスロットにないときは何も表示しません。〈表示位置〉〈カラーコード〉〈論理演算子〉についてはPUT　SPRITE、COLOR、LINEを参照してください。

DRAW
(ドロー)

機能　画面上に図形を描きます。

書式　DRAW "〈文字列〉"

解説

作図命令を使って、画面上に図形を描きます。作図命令は15種類あり、文字で表わされます。それらを組み合わせて図形を描きます。

ここで使われる作図命令には次のようなものがあります。

●移動命令

U〈距離〉　上へ移動
D〈距離〉　下へ移動
L〈距離〉　左へ移動
R〈距離〉　右へ移動
E〈距離〉　右斜め上へ移動
F〈距離〉　右斜め下へ移動
G〈距離〉　左斜め下へ移動
H〈距離〉　左斜め上へ移動

H　U　E
L　　　R
G　D　F

Mx, y　　移動位置をx方向、y方向各々についての絶対座標あるいは相対座標で指定します。ここで、xの前にプラス(＋)あるいはマイナス(－)の符号が書かれていれば、相対座標指定と解釈されます。

移動命令は、基準点からそれぞれの方向に指定された距離だけ作図します。各命令の実行後の基準点は描き終わった最後の点となります。各方向への移動距離は、上の各命令中の〈距離〉にS命令で決められる倍率をかけたものになります。

なお、各命令に同じ〈距離〉を指定しても、画面上の実際の距離は、上図のように作図方向により異ります。距離の単位は、ドットです。

●そのほかの命令

B　各々の方向へ移動しますが、作図は行ないません。

N　各々の方向へ作図しながら移動しますが、作図後の基準点は始点となります。

A〈角度〉　U、D、L、R、E、F、G、Hおよび相対座標指定のときのM命令で作図した図形を90°単位で反時計方向に回転させて表示します。一度指定すると次のA命令まで有効となります。ここで〈角度〉は0～3の数字で表わされ、0は0度、1は90度、2は180度、3は270度を表わします。省略すると0とみなします。

C〈カラーコード〉

図形に色をつけます。〈カラーコード〉は0～15の値を指定します。

省略すると、文字色と同じとみなします。

S〈倍率係数〉倍率を決めます。

〈倍率係数〉の範囲は0～255の整数です。この値の4分の1が倍率となります。たとえば〈数値〉に1を指定した場合、倍率は4分の1となります。ただし0は4と同じで、倍率1となります。U、D、L、R、E、F、G、H命令およびM命令の相対指定で指定された距離に倍率がかけ合わされて、移動・作図時の距離が決められます。〈倍率係数〉の指定を省いた場合は0が設定されます。S命令は指定すると、次のS命令まで有効となります。

X〈文字変数〉;　〈文字変数〉で示される作図命令を実行します。X命令を使用することにより、部分的な図形を定義することができます。

以上の命令中、〈距離〉や〈カラーコード〉などには、定数の他に「＝〈変数名〉;」の形式で数値変数を指定することができます。

サンプルプログラム

```
10  SCREEN 2
20  A$= "U80R80D80L80"
30  DRAW "BM80, 150XA$;"
40  GOTO  40
50  END
```

(80,　150)を1つの頂点とする1辺の長さ80の長方形を描きます。

音楽演奏ステートメント

BEEP
(ビープ)

➡ 基礎編 108ページ

機能　スピーカーから音を出します。

書式　BEEP

解説

PRINT　CHR$(7)と同じ機能を持ちます。約0.04秒間スピーカーを鳴らします。またSET　BEEPによって音色、音の大きさを変えることができます。
(SET参照)

PLAY
(プレイ)

➡ 基礎編108ページ

機能　音楽を演奏します。

書式　PLAY 〈文字式1〉 [，〈文字式2〉] [，〈文字式3〉]

文例　PLAY "CDEFGAB", "DEFGABC", "EFGABCD"
(「ドレミファソラシ」と「レミファソラシド」と「ミファソラシドレ」を同時に演奏します。)

解説

〈文字式１〉、〈文字式２〉、〈文字式３〉で指定された音楽を演奏します。３つの文字式で、３重の和音を出力することができます。
文字式に使われる記号の意味は次のとおりです。

i）A、B、C、D、E、F、G、♯、＋、－
現在指定されているオクターブで、音を出します。A～Gはそれぞれ

C	D	E	F	G	A	B
ド	レ	ミ	ファ	ソ	ラ	シ

を表わします。これらの文字の後に書かれた「♯」、「＋」、「－」はそれぞれ

♯、＋…シャープ(半音上げる)
－　…フラット(半音下げる)

を意味し、ピアノの黒い鍵盤に相当します。ただし、黒鍵に相当しないＣ－はB、E＋はF、F－はE、B＋はCと同オクターブの別のキーに対応します。

ii）O〈数字〉
Oの後に書いた〈数字〉でオクターブを指定します。１～８の数字で８オクターブ指定できます。Oが指定されないときは、以前のオクターブ値を保持します。電源を入れたときにはO4に設定されています。

iii）N〈数字〉
Nの後に書かれた〈数字〉でA～G、♯、＋、－の代わりに音階を指定します。０～96の数字が使えますが、0は休符を指定したことになります。O1Cの音はN〈数字〉ではあらわせません。

O１

C	C＋ D－	D	D＋ E－	E	F	F＋ G－	G	G＋ A－	A	A＋ B－	B
	1	2	3	4	5	6	7	8	9	10	11

O４

C	C＋ D－	D	D＋ E－	E	F	F＋ G－	G	G＋ A－	A	A＋ B－	B
36	37	38	39	40	41	42	43	44	45	46	47

iv）L〈数字〉
Lの後に書かれた〈数字〉で、次から演奏する音の長さを指定します。１～64の数字を使うことができます。電源投入時の値は４です。

L１：全音符
L２：２分音符
⋮
L４：４分音符
⋮
L８：８分音符
⋮
L64：64分音符

v）R〈数字〉
休符です。〈数字〉の数だけ演奏を休みます。〈数字〉の意味は「L」と同じです。

vi）・(ピリオド)
音の長さを表わす数字の後につけると、符点音符を意味するようになります。R(休符)の後につけることもできます。

vii）T〈数字〉
〈数字〉で曲全体のテンポを指定します。〈数字〉は１分間に演奏される４分音符の数です。32～255までが指定できます。電源投入時の値は120です。

viii）V〈数字〉
音の大きさを指定します。〈数字〉は０～15の範囲で、０は音を出しません。電源投入時の値は８です。〈数字〉は13くらいが適当です。

ix）S〈数字〉
出力波形の形状を指定します。〈数字〉は１～15が指定でき、次の図に示すような波形を出力します。電源投入時の値は１です。

Sの値	音の波形
0～3、9	
4～7、15	
8	
10	
11	
12	
13	
14	

指定を解除するときにはV<数字>を使ってください。もとの波形に戻ります。したがってVとSは同時に指定できません。M<数字>といっしょに用いて、音色を変化させるのに使います。

x）M<数字>

<数字>でエンベロープの周期(次ページ参照)を指定します。1～65535までの数字が設定できます。電源投入時の値は255です。

X<変数>;

文字変数をPLAY文中で使う場合、X<変数>;の形で入れることができます。

以上の命令中 <数字> のところは定数の他に、「=<変数名>;」の形式で数値変数を指定することができます。

SOUND
(サウンド)

機能　**指定したPSG(サウンドIC)のレジスタに直接データを書き込み、音を発生させます。**

書式　**SOUND <レジスタ番号>, <データ>**

解説

PSGのR0～R13まで14個のレジスタに直接データを書き込むことによって、PLAY命令では出すことのできない複雑な音を発生させることができます。

レジスタ＼ビット		B7	B6	B5	B4	B3	B2	B1	B0
R 0	チャンネルA周波数	8BIT				FT A			
R 1	チャンネルA周波数					4BIT CT A			
R 2	チャンネルB周波数	8BIT				FT B			
R 3						4BIT CT B			
R 4	チャンネルC周波数	8BIT				FT C			
R 5						4BIT CT C			
R 6	ノイズ周波数				5BIT NP				
R 7	チャンネル設定	IN/OUT		NOISE			TONE		
		1	0	C	B	A	C	B	A
R 8	チャンネルA音量				M	L3	L2	L1	L0
R 9	チャンネルB音量				M	L3	L2	L1	L0
R10	チャンネルC音量				M	L3	L2	L1	L0
R11	エンベロープ周期	8BIT FT							
R12		8BIT CT							
R13	エンベロープ形状					E3	E2	E1	E0

i）R0～R5

発生させたい音の周波数を設定します。PSGには3チャンネルのトーンジェネレータがあるので、R0、R1でチャンネルA、R2、R3でチャンネルB、R4、R5でチャンネルCを指定します。

ここではチャンネルAのR0、R1について説明します。出力したい周波数をf_Tとした場合

$$\frac{1.7898\times10^6}{16\times f_T}$$

の上位1バイトがR1に、下位1バイトがR0に入ります。

例）f_T＝200Hzの場合

$$\frac{1.7898\times10^6}{16\times200}\fallingdotseq559$$

559は&H22Fですから、R1には&H2、R0には&H2Fを書き込めばよいことになります。プログラムでは、

```
SOUND 1, &H2
SOUND 0, &H2F
```

を実行すればよいわけです。

R2～R5についても同様です。

ii）R6

ノイズ周波数を設定します。出力したいノイズ周波数をf_Nとすると、

$$\frac{1.7898\times10^6}{16\times f_N}$$

をR6に入れます。

例）f_N＝10kHzの場合

$$\frac{1.7898\times10^6}{16\times10\times10^3}\fallingdotseq11$$

従ってR6には11を書き込めばよいわけです。

プログラムでは

```
SOUND 6, 11
```

を実行します。

iii）R7

使用したいチャンネル(A、B、C)およびノイズ発生の有無を指定します。使用したいチャンネルのビットを0にしたデータを書き込むことで指定できます。

例）チャンネルAのトーンとA、Bのノイズを使う場合、

B7	B6	B5	B4	B3	B2	B1	B0
1	0	1	0	0	1	1	0

となるので&B10100110(&HA6)をR7に書き込めばよいわけです。プログラムでは

```
SOUND 7, &B10100110
```

を実行してください。

注） B7＝1、B6＝0と指定してください。

iv）R8～R10

A、B、C各チャンネルの音量を設定します。下位4ビットに0～15の数値を書き込むことによって音量を変えることができます。0のときが無音、15が最大です。

また、これらのレジスタのB4を1に設定すると音量が時間的に変化するモード(エンベロープ・モード)にすることができます。この場合、どのような形状でエンベロープを出力するのか、またその周期はどのくらいになるのかをR11～R13で指定しなければなりません。

振幅

エンベロープ周期

例）チャンネルAの音量を15に、Bの音量を7にし、チャンネルCをエンベロープ・モードで使いたい。

```
SOUND 8, 15
SOUND 9, 7
SOUND 10, 16
```

v）R11～R12

エンベロープ周期を設定します。エンベロープ周期をf_Eとするとき、

$$\frac{1.7898\times10^6}{256\times f_E}$$

の上位1バイトをR12に下位1バイトをR11に書き込みます。

例）$f_E=0.2$Hzの場合

$$\frac{1.7898\times10^6}{256\times0.2}\fallingdotseq35000$$

35000は&H88B8ですから、R12には&H88、R11には&HB8を書き込めばよいことになります。プログラムでは、

```
SOUND 12, &H88
SOUND 11, &HB8
```

を実行すればよいのです。

vi）R13

下位4ビットでエンベロープの形状を設定します。E0からE3の値がどのような波形に対応しているのかを下図に示します。なお、図中の×は0でも1でもかまいません。

E3	E2	E1	E0		対応する10進数
0	0	×	×		0～3
0	1	×	×		4～7
1	0	0	0		8
1	0	0	1		9
1	0	1	0		10
1	0	1	1		11
1	1	0	0		12
1	1	0	1		13
1	1	1	0		14
1	1	1	1		15

エンベロープ周期

例）上図の上から3番目の波形でエンベロープを出力したい。

```
SOUND 13, &B1000
```

サンプルプログラム

```
10  FORN=0 TO 15
20  SOUND 0, &H18
30  SOUND 1, &H1
40  SOUND 7, &HFE
50  SOUND 8, N
60  SOUND 11, &H46
70  SOUND 12, &H0
80  SOUND 13, &H8
90  FOR J=1 TO 100
100 NEXT J, N
110 END
```

(PSGのチャンネルAを使って音を発生させます。)

ファンクションキー関係ステートメント

KEY
(キー)

機能　ファンクションキーの内容を決めます。

書式　KEY 〈ファンクションキー番号〉，〈文字式〉

文例　KEY 6，"CLS"＋CHR$（13）
（ファンクションキーF6にCLS RETURN を定義します。）

解説

ファンクションキーに文字列を定義します。〈ファンクション キー番号〉は1から10までです。１つのキーには最大15文字の文字列が定義できます。16文字以上定義しようとしても、残りは無視されます。ただし画面表示は、５文字までです。

RETURN（キャラクターコード13）や、「"」（キャラクターコード34）はCHR$関数を使って定義します。（付録のコントロールコード一覧表を参照してください。）電源を切ると設定は初期状態に戻ります。

KEY LIST
(キー リスト)

機能　**ファンクションキーの定義内容を画面に表示します。**

書式　KEY LIST

解説

ファンクションキーに定義されている文字列を、F1からF10まで順に画面に表示します。

注）F10のRUNは、F5のRUNと違い、CLS命令のコントロールコードが入っているため、先頭にスペースが表示されます。

KEY ON/OFF
(キー オン/オフ)

機能　**ファンクションキーに定義される文字列を表示します。**

書式　①KEY ON
②KEY OFF

解説

①KEY ON

テキスト画面にファンクションキーの表示を行なっているため、この状態では最下行は使用できません。

電源投入時はこの状態です。

②KEY OFF

この命令を実行させるとファンクションキーの表示が消え、最下行も使用可能となります。

ファンクションキーからの入力はできます。

エラー処理ステートメント

ERROR
(エラー)

機能　エラーを模擬的に発生させます。

書式　ERROR <エラーコード>

文例　ERROR 1 (エラーコード1に対応するエラーを模擬的に発生させます。)

解説

<エラーコード>で指定したエラーが発生した状態を模擬的に作りだします。<エラーコード>は1～255の整数でなければなりません。
エラーコードのうち、26～49および60～255は、未定義のエラーコードでこれらの値を指定すると、「Unprintable error」を出力します。
エラーコードについては、付録のエラーメッセージ一覧表を参照してください。

ON ERROR GOTO
(オン エラー ゴートゥー)

機能　エラーが発生したとき、指定した行番号に実行を移します。

書式　ON ERROR GOTO <行番号>

文例　ON ERROR GOTO 100
(プログラムの実行中にエラーが発生した場合は、行番号100へ飛びなさい。)

解説
ON ERROR GOTO文を前もって実行しておくと、プログラム中で何かのエラーが生じたときに、指定した<行番号>に実行を移します。これにより、エラーの発生によるプログラムの実行の中断をふせぐことができます。
<行番号>に 0 を指定してメインプログラムで実行するか、またはCLEAR命令を実行すると以上の働きをなくします。エラー回復処理ルーチン中で実行すると、エラー回復処理を中止し、エラーメッセージを画面に表示してプログラムの実行を中断します。通常は、RESUMEで戻ります。

注) 命令待ちの状態で発生したエラー、およびエラー回復処理ルーチンを実行中に発生したエラーに対しては効力がありません。
また、INPUT文実行中にキー入力された内容によって発生するエラーは、再び入力待ちの状態となるため、エラー処理は行ないません。<行番号>に 0 を指定しても、ON ERROR GOTO 0 として動作するため、行番号 0 をエラー処理ルーチンの開始行に指定することはできません。

RESUME
(リジューム)

機能　エラー処理終了後、指定した場所から実行を再開します。

書式　① RESUME [<行番号>]
② RESUME NEXT

解説

エラー処理プログラムの実行後、メインプログラムの実行を再開します。実行を再開する場所に応じて３つの書式を選ぶことができます。

① RESUME 〈行番号〉

〈行番号〉の行から実行が再開されます。
ただし、〈行番号〉に０を指定した場合は、
エラーの生じた文から実行が再開されます。

② RESUME NEXT

エラーの生じた文の次の文から実行が再開されます。

③ RESUME

エラーの生じた文から実行が再開されます。

RESUME文はエラー処理プログラムの終わりを示すもので、メインプログラムへ復帰しない場合を除いて必ず置く必要があります。

サンプルプログラム

```
10  ON ERROR GOTO 100
20  A=6
30  FOR B=3 TO -3 STEP-1
40  C=A/B
50  PRINT A; "/ ";B; " = " ;C
60  NEXT B
70  END
100 PRINT A; "/ ";B; " =とけません"
110 RESUME 60
```

（エラー（０で割る）が発生した場合、100行目に飛び、「＝とけません」と表示して、60行目に戻ります。）

割り込み処理ステートメント

ON INTERVAL GOSUB
(オン インターバル ゴーサブ)

機能　行番号で指定したインターバルタイマ割り込みルーチンへ飛びます。

書式　ON INTERVAL＝〈時間〉 GOSUB 〈行番号〉

文例　ON INTERVAL＝60 GOSUB 100
(プログラム実行中、1秒ごとに行番号100のサブルーチンを実行します。)

解説
〈時間〉に書かれた数値の1/60秒ごとに、〈行番号〉で始まる割り込みルーチンを実行します。
この文を有効とするためには、以前にINTERVAL ON命令を実行しておく必要があります。
また割り込みルーチンでは、タイマ割り込みは受けつけなくなります。

注） ベーシック・プログラムの実行中でないと割り込みは発生しません。
また、ON ERROR文と同時に使用することはできません。

INTERVAL ON/OFF/STOP
(インターバル オン/オフ/ストップ)

機能　インターバルタイマによる割り込みの発生を許可、禁止、または停止します。

書式　①INTERVAL ON
②INTERVAL OFF
③INTERVAL STOP

解説

①INTERVAL ON
新しい命令文を実行するたびに、インターバルタイマをチェックし、指定された時間になると、ON INTERVAL GOSUB<行番号>で指定した割り込みを発生させます。

②INTERVAL OFF
割り込みの発生は禁止され、インターバルタイマのチェックも行いません。

③INTERVAL STOP
割り込みの発生は禁止されますが、インターバルタイマのチェックは行われ、記憶されますので、INTERVAL ONが実行すると、INTERVAL STOP命令実行中にインターバルタイマが指定された時間を経過していた場合には、すぐに割り込みを発生させます。

サンプルプログラム

```
10  ON INTERVAL=120 GOSUB 50
20  TIME=0
30  INTERVAL ON
40  GOTO 40
50  BEEP
60  PRINT TIME
70  RETURN
```

(2秒ごとに、ブザーを鳴らし、時間を60倍の秒単位で表示します。)

ON KEY GOSUB
(オン キー ゴーサブ)

機能　ファンクションキーの割り込みが発生したときに飛ぶ、処理サブルーチンの開始行番号を指定します。

書式　ON KEY GOSUB <行番号> [, <行番号>…]

文例　ON KEY GOSUB100, 200, 300
(F1キーが押されたら行番号100へ、F2キーなら200へ、F3キーなら300へ処理が移るよう設定します。)

解説

この文が有効となるためには、以前にKEY(n)ON命令が実行される必要があります。処理ルーチンでは、RETURN文が実行されるまでファンクションキーの割り込みは受けつけません。処理ルーチンから復帰すると、再び割り込みを受けつけるようになります。

ベーシックプログラムを実行中でないと、割り込みは発生しません。またON ERROR GOTO文と同時に使用することはできません。

KEY (n) ON/OFF/STOP
(キー(エヌ)オン/オフ/ストップ)

機能　**指定されたファンクションキーによる割り込みの発生を許可、禁止、または停止します。**

書式　**①KEY (<ファンクションキー番号>) ON**
②KEY (<ファンクションキー番号>) OFF
③KEY (<ファンクションキー番号>) STOP

解説

① KEY(<ファンクションキー番号>)ON
ON KEY GOSUB<行番号>でファンクションキー番号による割り込みルーチンが指定されている場合に、新しい命令文を実行するたびに、ファンクションキー番号の状態をチェックし、押されると、割り込みを発生させます。

②KEY(<ファンクションキー番号>) OFF
割り込みの発生は禁止され、ファンクションキーの状態のチェックも行いません。

③KEY (<ファンクションキー番号>) STOP
割り込みの発生は禁止されますが、ファンクションキーのチェックは行われ、その内容は記憶されています。

KEY(<ファンクションキー番号>)ONを実行すると、KEY (<ファンクションキー番号>)STOP 命令の実行中に、ファンクションキーが押されていた場合には、自動的に割り込みを発生させます。

サンプルプログラム

```
10  PRINT "F1, F2キーをおしてください"
20  ON KEY GOSUB 90, 100
30  KEY(1) ON
40  KEY(2) ON
50  GOTO 50
60  KEY(1) OFF
70  KEY(2) OFF
80  END
90  BEEP : RETURN 60
100 FOR I=1 TO 30 : BEEP : NEXT I
110 RETURN 70
```

(ファンクションキー[F1]が押されるとブザーを1回だけ鳴らし、[F2]キーが押されるとブザーを30回鳴らして、実行を終了します。)

ON SPRITE GOSUB
(オン スプライト ゴーサブ)

機能　スプライトどうしの衝突があったとき、指定した処理ルーチンへ飛びます。

書式　ON SPRITE GOSUB 〈行番号〉

文例　ON SPRITE GOSUB 1000
（スプライトどうしの衝突があったとき行番号1000へ飛びます。）

解説

スプライトどうしの衝突があったとき〈行番号〉で指定した割り込み処理ルーチンへサブルーチン・ジャンプします。
処理ルーチンの内部では、割り込みは受けつけられません。RETURN文が実行され、もとのルーチンへ戻ってきたときに、再び割り込みを受けつけるようになります。
また、割り込みはプログラムを実行中でないと発生しません。ON ERROR GOTO文と同時に使用することはできません。

SPRITE ON/OFF/STOP
(スプライト オン/オフ/ストップ)

機能　スプライトの衝突による割り込みを制御します。

書式
①SPRITE ON
②SPRITE OFF
③SPRITE STOP

解説

①SPRITE ON
スプライトの衝突による割り込みを可能にしたい場合、この命令を実行します。この命令を実行後、割り込みがあった場合、プログラムの流れは、ON SPRITE GOSUB文で指定した処理ルーチンへと移ります。
②SPRITE OFF
スプライトの衝突による割り込みが発生しないようにします。この命令を実行後は、次のSPRITE ON命令を実行するまではON SPRITE GOSUB文は何の意味も持ちません。
③SPRITE STOP
スプライトの衝突による割り込みを中止します。SPRITE OFFと違うのは、中止の間にスプライトが衝突したか否かを覚えているということです。中止の間に衝突があった場合、次にSPRITE ON命令が実行されれば、すぐに割り込みを発生します。

サンプルプログラム

```
10  SCREEN 2,0
20  FOR I=1 TO 8
30  READ A:B$=B$+CHR$(A)
40  NEXT I
50  SPRITE$(0)=B$
60  ON SPRITE GOSUB 140
70  SPRITE ON
80  FOR I=0 TO 192
90  PUT SPRITE 1,(50+I,10+I),1,0
```

```
100 PUT SPRITE 2, (220−I, 180−I), 8, 0
110 NEXT I
120 DATA 0, 16, 40, 124, 254, 84, 40, 0
130 END
140 SPRITE OFF
150 FOR J=1 TO 10: BEEP: NEXT J
160 RETURN 130
```

(スプライト面番号1と2のスプライトの衝突が発生すると、プログラムの実行が140行目に移り、ブザーを10回鳴らして実行を終了します。)

ON STOP GOSUB
(オン ストップ ゴーサブ)

機能　[CTRL] + [STOP] キーが押されて、割り込みが発生したときに飛ぶサブルーチンの開始行を指定します。

書式　ON STOP GOSUB〈行番号〉

文例　ON STOP GOSUB 200

([CTRL] + [STOP] キーが押されたら、行番号200の割り込み処理サブルーチンに飛びます。)

解説

[CTRL] + [STOP] キーが押されるたびに、〈行番号〉で始まる割り込みサブルーチンを実行します。

この文を有効とするためには、キーが押される以前に、STOP ON命令を実行しておく必要があります。

割り込みルーチンでは、[CTRL] + [STOP] キーによる割り込みは起こりません。

ベーシック・プログラムの実行中にのみ有効です。

エラーなどにより、プログラムの実行が停止している場合には、無効となります。

ON ERROR GOTO文と同時に使用することはできません。

注)

ON STOP GOSUB 〈行番号〉 には、プログラムの実行を停止する [CTRL] + [STOP] キーの操作が、命令の中に含まれますので、プログラムの実行が停止できなくなる可能性があります。次のようなプログラムを実行すると、電源を切る以外に方法はありません。

```
10  ON STOP GOSUB 40
20  STOP ON
30  GOTO 30
40  RETURN
```

STOP ON/OFF/STOP
(ストップ オン/オフ/ストップ)

機能　CTRL ＋ STOP キーによる割り込みの発生を許可、禁止または停止します。

書式
①STOP ON
②STOP OFF
③STOP STOP

解説

①STOP ON
新しい命令文を実行するたびに、CTRL + STOP キーの状態をチェックし、CTRL + STOP キーが押されると、ON STOP GOSUB〈行番号〉で指定した割り込みを発生させます。

②STOP OFF
割り込みの発生は禁止され、CTRL + STOP キーの状態のチェックを行いません。

③STOP STOP
割り込みの発生は禁止されますが、CTRL + STOP キーのチェックは行われ、記憶されますので、STOP ONを実行するとSTOP STOP命令実行中に CTRL + STOP キーが押されていた場合には、自動的に割り込みを発生させます。

サンプルプログラム

```
10  ON STOP GOSUB 70
20  STOP ON:CLS
30  FOR I=0 TO 500
40  LOCATE 3, 10 :PRINT"PUSH  CTRL&STOP
    KEYS";
50  NEXT I
60  END
70  PRINT I
80  RETURN 60
```

（CTRL + STOP キーが押されると、そのときのIの値を表示してプログラムの実行を終了します。）

ON STRIG GOSUB
(オン エストリガ ゴーサブ)

機能　ジョイスティックのトリガボタンが押されて、割り込みが発生したときに飛ぶサブルーチンの開始行を指定します。

書式　ON STRIG GOSUB〈行番号〉[,〈行番号〉,……]
〈行番号〉は最大4つ

文例　ON STRIG GOSUB 100, 200

解説

トリガボタンが押されるたびに、〈行番号〉で始まる割り込みサブルーチンを実行します。
この文を有効とするためには、トリガボタンが押される以前に STRIG ON 命令を実行しておく必要があります。
割り込みルーチンでは、トリガボタンによる割り込みは起こりません。
飛び先の行番号と、使用されるトリガボタンとの対応は、つぎのとおりです。

0……スペースバー
1，3……ジョイスティック1
2，4……ジョイスティック2

プログラムの実行中にのみ、この命令は有効です。
ON ERROR GOTO 文と同時に使用することはできません。

STRIG ON/OFF/STOP
(エストリガ オン/オフ/ストップ)

機能　ジョイスティックのトリガボタンによる割り込みを許可、禁示、または停止する。

書式 ①STRIG (〈ボタン番号〉) ON
②STRIG (〈ボタン番号〉) OFF
③STRIG (〈ボタン番号〉) STOP

解説

〈ボタン番号〉は使用されるトリガボタンの種類を指定します。〈ボタン番号〉の値は、0～4までで、使用されるトリガボタンとの対応はつぎのとおりです。

ボタン番号	使用されるトリガボタン
0	スペースバー（トリガボタンとして使用）
1または3	ジョイスティック1のトリガボタン
2または4	ジョイスティック2のトリガボタン

①STRIG 〈ボタン番号〉 ON
命令文が実行されるたびにトリガボタンが押されたかどうかをチェックします。トリガボタンが押されると、ON STRIG GOSUB 〈行番号〉による割り込みを発生させます。

②STRIG 〈ボタン番号〉 OFF
割り込みの発生を禁止し、トリガボタンが押されたかどうかのチェックも行いません。

③STRIG 〈ボタン番号〉 STOP
割り込みは禁止されますが、トリガボタンの状態はチェックされ、記憶されます。STRIG (〈ボタン番号〉) STOP命令の実行中にトリガボタンが押されていた場合には、STRIG (〈ボタン番号〉) ON命令を実行すると、自動的に割り込みを発生させます。

サンプルプログラム

```
10  ON STRIG GOSUB 60
20  STRIG (0) ON
30  PRINT "PUSH! SPACE BAR"
40  GOTO 40
50  END
60  BEEP
70  RETURN 50
```

（スペースバーを押すと、ブザーを鳴らし、プログラムの実行を終了します。）

特殊ステートメント

POKE
(ポーク)

➡ 基礎編135ページ

機能　**指定した番地のメモリにデータを書き込みます。**

書式　**POKE 〈メモリの番地〉, 〈データ〉**

文例　**POKE　&HD000, 20**

（メモリの&HD000番地に数値20を書き込みます。）

解説

〈メモリの番地〉は－32768～65535(&H0～&HFFFF)、〈データ〉は0～255の範囲になければなりません。いずれも小数部分は切り捨てられます。

〈メモリの番地〉が負の数の場合、65536を加えたものが実際の値になります。たとえばPOKE－1, 255はPOKE 65535, 255と同じです。

なお番地&HF380以降は、ベーシックが使用していますので、指定できません。

VPOKE
（ブイポーク）

機能　VRAM内の指定したアドレスにデータを書き込みます。

書式　VPOKE 〈VRAMのアドレス〉，〈データ〉

解説

VRAM（ビデオRAM）は画面に表示される文字や図形を記憶しているメモリです。このメモリに書き込まれたデータがビデオ信号に変えられて表示されます。

〈VRAMのアドレス〉で指定したVRAMに〈データ〉で指定した数値を書き込みます。〈VRAMのアドレス〉は0～65535（&H0～&HFFFF）、データは0～255(&H0～&HFF)の範囲になります。

画面モードは5～8の場合には＜VRAMのアドレス＞にアクティブページの開始アドレスを加えた値がVRAMの開始アドレスになります。

注） VRAMのマッピング、各パターン・テーブルの位置（BASE命令を参照してください。）などをよく理解した上でこの命令を使うようにしてください。

DEF USR
（デファイン ユーザ）

機能　機械語サブルーチンの開始番地を定義します。

書式　DEF　USR［〈番号〉］＝〈開始番地〉

文例　DEF　USR 1 ＝&HF000

（機械語ユーザ関数USR1の実行開始番地を&HF000に設定します。）

解説

機械語ユーザ関数USR0～USR9が呼び出す機械語サブルーチンの実行開始番地を設定します。〈番号〉は0～9で、省略すると「0」とみなします。

また、DEF　USR文は2重に定義してもエラーにはなりません。後から定義した方が有効になります。

OUT
（アウト）

機能　**出力ポートに１バイトのデータを送ります。**

書式　**OUT 〈ポート番号〉，〈式〉**

文例　**OUT &HFF，128**
(CPUのI/O出力ポート255番に数値128を出力します。)

解説

指定した〈ポート番号〉のCPUのI/O出力ポートに〈式〉の値を出力します。〈ポート番号〉および〈式〉の値は0～255(&H0～&HFF)の範囲になければいけません。
ただし、〈ポート番号〉が255を超えても、エラーは出ません。
通常この命令を使うことはないでしょう。

WAIT
（ウエイト）

機能　**指定した入力ポートから特定のデータを読み込む間、プログラムの実行を停止します。**

書式　**WAIT 〈ポート番号〉,〈式１〉［，〈式２〉］**

文例　**WAIT 255，1**
(ポート番号255からのデータの最下位ビットが１になるまで実行を停止します。)

解説

〈ポート番号〉で指定したポートから読み込まれたデータを〈式2〉とEXCLUSIVE ORし、さらに〈式１〉とANDした結果が０の間プログラムの実行を停止し、ポートから読み込みを続けます。そして、０でなくなれば次の命令から実行を再開します。
〈式１〉、〈式２〉は値が整数(0～255)でなければいけません。
〈式２〉が省略された場合、その値は０とみなします。

CALL
(コール)

機能　ROMカートリッジで用意された拡張ステートメントやRAMディスクの操作命令、プリンタ出力命令を呼び出します。

書式　CALL <拡張ステートメント名> [(<式>, <式>, ……)]

文例1　CALL　MEMINI (<RAMディスクの上限>)
(RAMディスクで使うメモリの上限を指定します。)
範囲は、&H0〜&H7FFFです。初期状態では、&H7FFFが指定されています。

文例2　CALL　MFILES
(RAMディスク中のファイル名を表示します。)

文例3　CALL　MKILL (<ファイル名>)
(RAMディスク中のファイルを削除します。)

文例4　CALL　MNAME (<旧ファイル名>) AS <新ファイル名>)
(RAMディスク中のファイルを付け替えます。)

解説
省略形として「_」(アンダーライン) を使用することができます。この命令に関する詳しい説明は各ROMカートリッジの説明書または、RAMディスクの操作命令、プリンタ出力命令の項をご覧ください。

注) CLEAR命令で設定したメモリの上限値が小さすぎるとCALL文が使用できなくなる場合がありますのでご注意ください。このようなときは文字領域の大きさとメモリの上限値を再設定してください。なお詳しくは135ページのCLEAR命令をご覧ください。

(例) CLEAR　200 , &HF37F

MOTOR ON/OFF
(モータ オン/オフ)

機能　カセットテープレコーダのリールモータのオン/オフをコントロールします。

書式①MOTOR
②MOTOR ON
③MOTOR OFF

解説

①MOTOR

カセットレコーダのリール モータが現在動いていればオフに、止まっていれば、オンにします。

②MOTOR ON

カセットレコーダのリールモータはオンになり、動きます。

③MOTOR OFF

カセットレコーダのリールモータはオフになり、一時停止状態になります。

COPY
(コピー)

機能　VRAM、配列変数、ディスクファイルの間で画像データを転送します。

書式1　COPY（転送元の座標１）－（転送元の座標２）［，＜転送元のページ＞］
TO（転送先の座標）［，＜転送先のページ＞］［，＜論理演算子＞］

書式2　COPY（転送元の座標１）－（転送元の座標２）［，＜転送元のページ＞］
TO＜配列変数｜ファイル名＞

書式3　COPY＜配列変数｜ファイル名＞［，＜方向＞］TO（転送先の座標）
［，＜転送先のページ＞］［，＜論理演算子＞］

COPY＜ファイル名＞TO＜配列変数＞

COPY＜配列変数＞TO＜ファイル名＞

解説

画面モードが5から8の時に有効です。

転送するデータは、転送元の座標1（X1，Y1）と転送元の座標2（X2，Y2）の長方形の領域をLINE命令と同じ方法で指定します。

転送元のページ、転送先のページを省略するとアクティブページが選択されます。

論理演算子には、AND 、OR 、XOR 、PSET PRESETが使用できます。

転送は転送元の座標1から始まり、転送元の座標2で終わります。従って、同じ領域を指定しても転送の向きが4通り存在します。＜方向＞はこの転送の向きを次の0から3の数値で指定します。

0　左上から右下
1　右上から左下
2　左下から右上
3　右下から左上

方向を省略すると0が選択されます。

配列変数は数値型の配列変数で、画像データを格納するために必要な大きさを持っていなければなりません。

画像データは次の方法で計算が可能です。

INT((〈ピクセルサイズ〉＊(ABS (X2－X1)＋1)
＊ (ABS (Y2－Y1)＋1)＋7) ／8)＋4バイト

ピクセルサイズ

SCREEN　5,7→4
SCREEN　6→2
SCREEN　8→8

SET
（セット）

機能　**電源ONのときに自動設定したい色々な項目をCMOS（時計IC）にセットします。**

書式　**SET〈セットする項目名〉［〈引数〉［，〈引数〉……］］**

書式1　**SET　ADJUST（X，Y）**

テレビモニターの表示を上下左右に調整します。

X，Yはそれぞれ－7～＋8の値です。

書式2　**SET　BEEP〈音色〉，〈音の大きさ〉**

BEEP音を設定します。

〈音色〉，〈音の大きさ〉はそれぞれ1～4の値です。

書式3　**SET　DATE"〈日付〉"［，A］**

〈日付〉を現在の日付としてセットします。〈日付〉は85／04／25のように年月日をスラッシュで区切ってください。

オプションのAを付けるとセットされた日付にアラームが設定されます。

書式4　**SET　PAGE〈ディスプレイページ〉，〈アクティブページ〉**

VRAM上でページングを行います。

〈アクティブページ〉は書き込みや読み込が行われるページで〈ディスプレイページ〉は表示されるページです。

指定できるページ数はスクリーンモードとVRAMの容量によって異なります。(次ページ参照)

書式5　**SET　PASSWORD"〈パスワード〉"**

電源ONのとき〈パスワード〉を入力しないとそのシステムが作動しないようにします。

〈パスワード〉は255文字以内です。なお設定した、パスワードを忘れてしまった場合には、GRAPHキーとSTOPキーを押しながらリセットしてください。

書式6　**SET　PROMPT"〈プロンプト〉"**

BASICのプロンプト"OK"を好きなように変更します。

〈プロンプト〉は6文字以内です。

書式7　**SET　SCREEN**

現在のSCREENの各パラメータを初期値として登録します。

書式8　**SET　TIME"〈時刻〉"［，A］**

〈時刻〉を現在の時刻としてセットします。〈時刻〉は11：26：21のように時分秒をコロンで区切ってください。

オプションのAを付けるとセットされた時刻にアラームが設定されます。

書式 9　SET　TITLE"〈タイトル〉"［，〈色の番号〉］

タイトルとその色を指定します。

〈タイトル〉は初期画面に6文字以内で表示されます。ちょうど6文字のときは何かキーを押すまで初期画面のままになります。

色の番号は1～4までの範囲です。（カラーパレットのナンバーとは関係がありません。）

注） 書式3，5，9の機能は一緒に使うことはできません。

この中で2以上を指定すると後で指定した項目だけが有効となります。

書式10　SET　VIDEO〈モード〉［，〈TV画面密度〉］［，〈VDPカラーバス〉］［，〈同期〉］［，〈音声〉］［，〈ビデオ入力〉［，〈AVコントロール〉］

スーパーインポーズ・音声混合・ビデオ入力の切り換え・AVコントロールのON／OFFなどのモードを設定します。

〈モード〉は以下のようになります。

0：内部同期・コンピュータ画面
1：外部同期・コンピュータ画面
2：外部同期・スーパーインポーズ
3：外部同期・外部ビデオ画面

〈TV密度〉は0でノーマル、1でハーフトーンになります。
〈VDPカラーバス〉は0で出力モード、1で入力モードになります。
〈同期〉は0で内部と、1で外部と同期します。
〈音声〉は以下のようになります。

0：音声混合をしない。
1：左チャンネルを混合する。
2：右チャンネルを混合する。
3：両チャンネルを混合する。

〈ビデオ入力〉は0でRGBコネクタから、1でRCAコネクタからビデオ入力をします。
〈AVコントロール〉はAVコントロール端子の出力を0でOFF、1でONにします。

VRAM容量とページ数

VRAM容量 ＼ 画面モード	5	6	7	8
64KB	2	2	—	—
128KB	4	4	2	2

GET
（ゲット）

機能　文字変数に現在の日付・時刻を代入します。

書式1　GET　DATE〈文字変数〉［，A］

書式2　GET　TIME〈文字変数〉［，A］

解説

GET　DATE〈文字変数〉［,A］
（文字変数）に現在の日付を代入します。
オプションのAを付けると現在セットされているアラームの日付が代入されます。

GET　TIME〈文字変数〉［,A］
〈文字変数〉に現在の時刻を代入します。
オプションのAを付けると現在セットされているアラームの時刻が代入されます。

数値関数

ABS関数
（アブソリュート）

➡ 基礎編56ページ

機能　**絶対値を与えます。**

書式　ABS（〈引数〉）

解説
〈引数〉の絶対値を与えます。
ABS(×)≡｜X｜

サンプルプログラム

```
PRINT ABS(−10) RETURN
```

（−10の絶対値を与えます。）

FIX関数
（フィックス）

機能　**整数を与えます。**

書式　FIX（〈引数〉）

解説

引数の整数部分を与えます。つまりSGN(X)*INT(ABS(X))と同じになります。
INT関数とFIX関数との大きな違いは、引数が負の場合、FIX関数は単にその整数部分に「－」をつけたものを返すという点にあります。

サンプルプログラム

```
10  A=-3.14
20  PRINT FIX(A)
30  PRINT INT(A)
40  END
```

(FIX関数とINT関数の〈引数〉に－3.14を入れた結果を表示します。)

INT関数
(インテジャー)

➡ 基礎編57ページ

機能　整数化を行ないます。

書式　INT (〈引数〉)

解説

結果は〈引数〉を超えない最大の整数となります。
INT(X)≡〔X〕

サンプルプログラム

```
PRINT INT (3.14) RETURN
```

(3.14の小数点部分を切り捨てて整数3を表示します。)

RND関数
(ランダム)

➡ 基礎編68ページ

機能　0から1の間の乱数を発生します。

書式　RND (〈引数〉)

解説

0より大きく、1よりも小さい乱数を発生します。
〈引数〉の値が正の場合は同一乱数系列の次の乱数を発生します。〈引数〉が0の場合は1つ前に発生した乱数と同じ数を与えます。〈引数〉が負の場合は引数の値によって決まる乱数を発生します。

サンプルプログラム

```
10  FOR I=1 TO  5
20  A=RND (1)
30  PRINT A
40  NEXT I
50  END
```

(乱数を5個発生させ、画面に表示します。)

SGN関数
（サイン）

➡ 基礎編57ページ

機能　符号を与えます。

書式　SGN（〈引数〉）

解説

〈引数〉が正ならば1を、〈引数〉が0ならば0を、〈引数〉が負ならば−1を与えます。

サンプルプログラム

PRINT SGN（−5）RETURN

（引数−5の符号を与えて、画面に出力します。）

SIN関数
（サイン）

機能　正弦（サイン）を与えます。

書式　SIN（〈引数〉）

解説

〈引数〉の正弦を与えます。〈引数〉の単位はラジアンです。

SIN(X)≡sin X

サンプルプログラム

PRINT SIN（3.14159/6）RETURN

（SIN $\pi/6$を、画面に表示します。）

COS関数（コサイン）

機能　余弦（コサイン）を与えます。

書式　COS（〈引数〉）

解説

〈引数〉の余弦を与えます。単位はラジアンです。

$\mathrm{COS}(X) \equiv \cos X$

サンプルプログラム

```
PRINT COS (3.14159/3) RETURN
```

（COS $\pi/3$ を画面に表示します。）

TAN関数（タンジェント）

機能　正接（タンジェント）を与えます。

書式　TAN（〈引数〉）

解説

〈引数〉の正接を与えます。〈引数〉の単位はラジアンです。

$\mathrm{TAN}(X) \equiv \tan X$

サンプルプログラム

```
PRINT TAN (3.14159/4) RETURN
```

（TAN $\pi/4$ を画面に表示します。）

ATN関数（アーク タンジェント）

機能　逆正接（アークタンジェント）を与えます。

書式　ATN（〈引数〉）

解説

〈引数〉の逆正接をラジアンで与えます。逆正接は正接(tan タンジェント)の逆三角関数です。与えられる数値範囲は、$-\pi/2$から$\pi/2$までです。

$\mathrm{ATN}(x) \equiv \tan^{-1} X$

サンプルプログラム

```
PRINT 4*ATN (1) RETURN
```

（$\mathrm{TAN}^{-1}(1)*4$ を画面に表示します。）

SQR関数
（スクエア　ルート）

機能　平方根を与えます。

書式　SQR（〈引数〉）

解説

〈引数〉の平方根を与えます。〈引数〉は0以上でなければなりません。

SQR(X) ≡ $\sqrt{X}$

サンプルプログラム

PRINT SQR (3) RETURN

（$\sqrt{3}$の値を画面に出力します。）

EXP関数
（エクスポネンシャル）

機能　eのべき乗を与えます。

書式　EXP（〈引数〉）

解説

eの〈引数〉乗を与えます。eは自然対数の底です。

EXP(X) ≡ e^X

引数の値は、145.06286085862以下でなければなりません。

サンプルプログラム

PRINT EXP (1) RETURN

（eの値を画面に表示します。）

LOG関数
(ログ)

機能　**自然対数を与えます。**

書式　LOG (〈引数〉)

解説

〈引数〉の自然対数を与えます。〈引数〉は正の値でなければいけません。

$LOG(X) \equiv \log_e X$

サンプルプログラム

PRINT LOG (2) RETURN

(2の自然対数を画面に出力します。)

CINT関数
(シー インテジャー)

機能　**数値を整数型に変換します。**

書式　CINT (〈引数〉)

解説

〈引数〉の値を整数型に変換します。〈引数〉の値は−32768〜32767の範囲になければなりません。

なお、小数部分は切り捨てられます。

サンプルプログラム

PRINT CINT (290.29) RETURN

(引数290.29を整数型に変えて画面に表示します。)

CDBL関数
(シー ダブル)

機能　数値を倍精度型に変換します。

書式　CDBL (<引数>)

解説

<引数>の値を倍精度型に変換します。単精度型、整数型の変数を精度よく計算したいときなどに用います。

サンプルプログラム

PRINT CDBL (1.23E+10) RETURN

(単精度型定数1.23E+10を倍精度型に変換して表示します。結果は12300000000となります。)

CSNG関数
(シー シングル)

機能　数値を単精度型に変換します。

書式　CSNG (<引数>)

解説

<引数>の値を7桁目で四捨五入して単精度型に変換します。メモリの節約にはなりますが、精度が落ちることに注意してください。

サンプルプログラム

PRINT CSNG (1.23456789) RETURN

(引数1.23456789を単精度型に変えて、画面に表示します。)

文字関数

LEFT$関数
(レフト ドル)

➡ 基礎編74ページ

機能　**文字列の左側から指定した長さの文字列を与えます。**

書式　LEFT$ (〈文字式〉, 〈引数〉)

解説

〈文字式〉の左から〈引数〉の数だけの文字で構成される文字列を与えます。〈引数〉の範囲は0から255までで小数部分が切り捨てられてから関数が評価されます。〈引数〉が0の場合、関数は、「" "」(ヌルストリング)を与えます。また、〈引数〉が、〈文字式〉の文字数より大きければ、関数は〈文字式〉すべてを与えます。

サンプルプログラム

```
PRINT LEFT$
("MICROCOMPUTER", 5) RETURN
```

(文字列 "MICROCOMPUTER" の左端から5文字目までの値を与えます。結果は "MICRO" となります。)

RIGHT$関数
(ライト ドル)

➡ 基礎編75ページ

機能　文字列の右側から指定した長さの文字列を与えます。

書式　RIGHT$ (〈文字式〉, 〈引数〉)

解説

〈文字式〉の右側から〈引数〉で指定した数の文字列を与えます。〈引数〉の範囲は0から255までで、小数部分が切り捨てられてから関数が評価されます。〈引数〉が0のとき、関数は「" "」(ヌルストリング)を与えます。また、〈文字式〉を構成する文字数より〈引数〉が大きい場合は、〈文字式〉すべてを与えます。

サンプルプログラム

```
PRINT RIGHT$
("PERSONALCOMPUTER", 8) RETURN
```

(文字列"PERSONALCOMPUTER"中の右から8文字目までを与えます。結果は、"COMPUTER"となります。)

MID$関数
(ミドル ドル)

➡ 基礎編75ページ

機能　文字列の中から指定した長さの文字列を抜き出します。

書式　MID$ (〈文字式〉, 〈引数1〉 [, 〈引数2〉])

解説

〈文字式〉において、左から〈引数1〉番目の文字から、〈引数2〉で指定した文字数の文字列を抜き出し与えます。〈引数1〉は1～255、〈引数2〉は0～255の範囲で指定します。また小数部分を切り捨ててから関数を評価します。〈引数2〉が0の場合および〈文字式〉の長さよりも大きい場合は、「" "」(ヌルストリング)を与えます。〈引数2〉を省略した場合，および〈文字式〉の〈引数1〉文字目以降の長さよりも〈引数2〉の方が大きい場合は、〈引数1〉文字目以降のすべてを与えます。

サンプルプログラム

```
PRINT MID$
("You have a computer." 5, 4) RETURN
```

(文字列"You have a computer."中の、左から5番目のhから4文字を抜き出します。結果は、"have"となります。)

LEN関数
(レングス)

➡ 基礎編76ページ

機能　文字列の文字数を与えます。

書式　LEN (<文字式>)

解説
<文字式>を構成している文字の数を与えます。このとき、画面に出力されない文字（スペースやキャラクターコードの1から31まで）も数えます。

サンプルプログラム
```
PRINT LEN ("MICROCOMPUTER") RETURN
```
(文字列“MICROCOMPUTER”の文字数を与えます。結果は、13となります。)

ASC関数
(アスキー)

➡ 基礎編118ページ

機能　キャラクターコードを与えます。

書式　ASC (<文字式>)

解説
<文字式>の先頭の文字のキャラクターコードを与えます。キャラクターコードと文字の対応は、「キャラクター コード表」を参照してください。

注) <文字式>が、「” ”」(ヌルストリング)のときには、Illegal function call エラーになります。

サンプルプログラム
```
PRINT ASC("Y")
```
(Yのキャラクターコードを画面に表示します。)

CHR$関数
(キャラクター ドル)

➡ 基礎編116ページ

機能　キャラクターコードに対応する文字を与えます。

書式　CHR$ (<引数>)

解説
<引数>をキャラクターコードとする文字を与えます。<引数>の範囲は0から255までで、小数部分が切り捨てられてから、関数が評価されます。キャラクターコードについては、「キャラクターコード表」を参照してください。

サンプルプログラム
```
PRINT CHR$(&H4F)
```
(キャラクターコード&H4Fに対応する文字を画面に表示します。)

VAL関数
（バリュー）

➡ 基礎編113ページ

機能　**文字列を数値に変換します。**

書式　VAL（〈文字式〉）

解説

〈文字式〉の表す数値を与えます。〈文字式〉の最初の文字が「＋」、「－」、「&」、数字のいずれでもなければ0を与えます。数字（16進数の場合はA～Fの英字も含む。先頭に&Hをつけます。）以外の文字が現われると、それ以降の文字はすべて無視されます。〈文字式〉中の空白はすべて無視されます。

サンプルプログラム

```
10  A$="12":B$="34"
20  C$=A$+B$
30  C=VAL (A$)+VAL (B$)
40  PRINT C, C$
50  END
```

（文字列12と34を、行番号20でつなぎ、数値12＋34の値を行番号30で計算して画面に表示します。）

STR$関数
（ストリング ドル）

➡ 基礎編114ページ

機能　**数値を文字列（数字）に変換します。**

書式　STR$（〈引数〉）

解説

〈引数〉の数値を表わす文字列を与えます。

サンプルプログラム

```
10  A=12:B=34
20  C=A+B
30  C$=STR$(A)+STR$(B)
40  PRINT C,C$
50  END
```

（数値12＋34の値と、文字列12と34をつないだ値を画面に表示します。）

STRING$関数
(ストリング ドル)

機能　1文字を指定した数だけ繰り返した文字列を与える。

書式　① STRING$ (〈繰り返しの数〉,〈キャラクターコード〉)
　　　　② STRING$ (〈繰り返しの数〉,〈文字式〉)

解説

①の場合、キャラクターコードに対応する文字列を〈繰り返しの数〉だけ連ねた文字列を与えます。
②の場合、〈文字式〉の値の先頭文字を〈繰り返しの数〉だけ連ねた文字列を与えます。
〈繰り返しの数〉、〈キャラクターコード〉はともに0～255の範囲になければいけません。ただし、電源投入時、〈繰り返しの数〉は、0～200の範囲で、これを変更する場合には、CLEAR命令を用います。
小数部分は切り捨てられます。

サンプルプログラム

```
PRINT STRING$ (5, "*") RETURN
```

(＊を5個表示します。)

SPACE$関数
(スペース ドル)

機能　指定した数だけ空白を与えます。

書式　SPACE$ (〈引数〉)

解説

〈引数〉個の空白からなる文字列を与えます。〈引数〉の値は0から255まで許されます。
ただし、電源投入時は、0～200の範囲で、これを変更する場合には、CLEAR命令を用います。
小数部分は切り捨てられます。

サンプルプログラム

```
PRINT SPACE$ (5) ; "K" RETURN
```

(画面左から5文字分の空白の後、Kを表示します。)

INSTR関数
(インストリング)

機能　**文字列を検索します。**

書式　INSTR ([〈引数〉,] 〈文字式1〉, 〈文字式2〉)

解説

〈文字式1〉の中の〈引数〉文字目（省略すると1文字目）以降からの〈文字式2〉を検索し、先頭文字から何番目にあるかを与えます。〈引数〉は0～255の範囲で指定でき、小数点以下があれば切り捨てられます。

〈引数〉が〈文字式1〉の長さより大きい場合や、〈文字式1〉が「" "」(ヌルストリング）の場合、または〈文字式2〉が〈文字式1〉の中に見つからない場合は0を与えます。〈文字式2〉がヌルストリングの場合、〈引数〉(省略したときは1)の値をそのまま与えます。

サンプルプログラム

```
10  B=0 : A$= "CENTIMETER"
20  B=INSTR (B+1, A$, "E")
30  PRINT B ;
40  IF B>0 THEN 20
50  END
```

(文字変数A$＝"CENTIMETER"の中に、"E"という文字が先頭(B＋1)から何番目にあるかをさがして画面に表示します。)

INKEY$関数
(インキー ドル)

➡ 基礎編77ページ

機能　**キーが押されていれば、その文字を、キーが押されていなければ、「" "」(ヌルストリング)を与えます。**

書式　INKEY$

解説

CTRL + STOP 以外のキーが押されていれば、その文字を、キーが押されていなければ、「" "」(ヌルストリング)を与えます。キーが押されるまで、待つ場合にはINPUT$関数が便利です。

サンプルプログラム

```
10  A$=INKEY$
20  IF A$=" " THEN GOTO 10
30  PRINT A$
40  END
```

(キーボードから押された文字を画面に表示します。)

INPUT$関数
(インプット ドル)

機能　**キーボードから指定された文字数の文字列を入力します。**

書式　INPUT$ (<引数>)

解説

キーボードから<引数>で指定した文字数の文字列を入力します。キーボードから入力されるキャラクターは、プログラムを中断する CTRL + STOP を除いて全て読み込まれます。

<引数>の値は1～255まで指定できます。ただし、201以上の値を使用するときにはCLEAR命令で文字領域の大きさを設定しておく必要があります。

また、引数の小数部分は切り捨てられます。

サンプルプログラム

```
10  A$=INPUT$(7)
20  PRINT  "HITACHI"+A$
30  END
```

(キーボードから適当な文字を7文字入力すると、HITACHIのすぐ後に続いて、入力した文字が表示されます。)

BIN$関数
(バイナリー ドル)

機能　**数値を2進数に変えた結果を文字列で与えます。**

書式　BIN$ (<引数>)

解説

<引数>を2進数で表す文字列に変換します。<引数> は－32768～65535の範囲になければなりません。<引数>が負の場合、それに65536を加えたものが、実際の引数となります。

また、小数部分は切り捨てられます。

サンプルプログラム

```
PRINT BIN$ (6305) RETURN
```

(6305を2進数に変換して表示します。)

OCT$関数
（オクト ドル）

機能　**数値を 8 進数に変えた結果を文字列で与えます。**

書式　OCT$（〈引数〉）

解説

引数を8進数で表わす文字列に変換します。〈引数〉は－32768～65535の間で指定します。
〈引数〉が負の場合、それに 65536 を加えたものが、実際の引数となります。

サンプルプログラム

```
PRINT OCT$ (56) RETURN
```

(10進数(56)を、 8 進数に変えて表示します。

HEX$関数
（ヘキサ ドル）

機能　**数値を16進数に変えた結果を文字列で与えます。**

書式　HEX$（〈引数〉）

解説

〈引数〉を16進数で表わす文字列に変換します。〈引数〉は－32768～65535の間の値で指定します。
〈引数〉が負の場合、それに 65536 を加えたものが、実際の引数となります。

サンプルプログラム

```
PRINT HEX$ (255) RETURN
```

(10進数(255)を16進数に変えて表示します。)

特殊関数

PEEK関数
(ピーク)

➡ 基礎編135ページ

機能　メモリ番地の内容を与えます。

書式　PEEK (〈メモリ番地〉)

解説

〈メモリ番地〉の内容１バイトを与えます。〈メモリ番地〉は式の形で指定でき、範囲は、−32768〜65535で小数部分が切り捨てられた後、実行されます。指定した番地が負の数の場合には、65536を加えたものが実際の値になります。

サンプルプログラム

```
PRINT PEEK (256) RETURN
```

(メモリ番地256の内容を表示します。)

VPEEK関数
(ブイ ピーク)

機能　VRAM内の指定したアドレスの内容を与えます。

書式　VPEEK (〈アドレス〉)

文例　PRINT　VPEEK(&H1000)

(VRAMアドレスの&H1000番地の内容を表示します。)

解説

〈アドレス〉で指定したVRAMの内容１バイト（0〜255）を与えます。〈アドレス〉は0〜65535（&H0〜&HFFFF）の範囲になければなりません。

USR関数
(ユーザ)

機能　ユーザが定義した機械語サブルーチンを呼び出します。

書式　USR［〈番号〉］(〈引数〉)

解説

〈引数〉を機械語サブルーチンに渡して、サブルーチンを呼び出します。〈番号〉は0から9までで、DEFUSR文で定義した番号に対応します。また、〈番号〉を省略すると「0」になります。

ベーシックから機械語への引数の引き渡しは、メモリを用いて行ないます。引数の型に従って以下の4つの場合があります。

i）整数型　&HF7F8、&HF7F9番地に引数が格納されます。&HF7F8番地に下位バイトがはいるので注意してください。また&HF663番地に、整数型を示す数値「2」が格納されます。

ii）単精度型　&HF7F6～&HF7F9番地の4バイトに引数が格納されます。&HF663番地には「4」がはいります。

iii）倍精度型　&HF7F6～&HF7FD番地の8バイトに引数が格納されます。&HF663番地には「8」がはいります。

iv）文字型　&HF7F8～&HF7F9番地の2バイトに、ストリング・ディスクリプタの番地が格納されます。ストリング・ディスクリプタは3バイトよりなり、先頭は文字の長さを示し、以下の2バイトで実際に文字のある番地を示しています。
&HF663番地には「3」が入ります。

VARPTR関数
(バー ポインター)

機能　変数のデータが格納されている先頭番地を与えます。

書式　① VARPTR (<変数名>)
　　　② VARPTR (#<ファイル番号>)

解説

①では変数や配列の要素のデータの格納されている先頭番地を与えます。<変数名>は変数名または配列の要素で指定します。
VARPTR関数実行の前に、その変数には、値が代入されていなければなりません。
またVARPTRの返す値は、−32768〜32767です。負の数のアドレスが返された場合、それに65536を加えた値が、実際のアドレスとなります。
②では、指定されたファイルを制御する部分の先頭番地を与えます。

サンプルプログラム

```
10  A=3
20  ADRS=VARPTR(A)
30  PRINT "a="; ADRS
40  END
```

(変数Aのデータが格納されている番地を画面に表示します。)

INP
(インプット)

機能　入力ポートから1バイトのデータを読み込みます。

書式　INP (<ポート番号>)

解説

指定した<ポート番号>の入力ポートから1バイトのデータを読み込みます。<ポート番号>の値は0〜255(&H0〜&HFF)の範囲でなければいけません。

サンプルプログラム

```
PRINT INP (&HFF) RETURN
```

(ポート番号255番から1バイトのデータを読み込みます。)

TAB関数
(タブ)

➡ 基礎編47ページ

機能　**指定した水平位置までカーソルを移動します。**

書式　TAB (〈引数〉)

解説

PRINTまたは、LPRINT文中で用いられ、〈引数〉で指定される水平位置までスペースを出力しながらカーソルを移動します。〈引数〉は0以上255以下の数値で、与えた〈引数〉が画面1行の表示文字数を超えていたら、1行の表示文字数で引いた値が用いられます。

サンプルプログラム

PRINT TAB (10) ; "A" RETURN

(画面水平位置10までカーソルを移動させます)

SPC関数
(スペース)

➡ 基礎編46ページ

機能　**指定された個数の空白を出力します。**

書式　SPC (〈引数〉)

解説

引数の数の空白を出力します。PRINT, LPRINT, PRINT＃文の中でのみ使用することができます。
引数の値は0～255の間で指定します。

サンプルプログラム

PRINT SPC (5) ; "A" RETURN

(空白を5つ画面に出力します)

FRE関数
（フリー）

機能　未使用領域を与えます。

書式　FRE（〈引数〉）

解説

① FRE(0)
〈引数〉が数式の場合はメモリの未使用領域の大きさをバイト数で与えます。

② FRE(” ”)
〈引数〉が文字式の場合にはメモリの文字領域のうち、使っていない領域の大きさをバイト数で与えます。

サンプルプログラム

```
10   CLEAR  300, &HE000
20   PRINT FRE(0) ; FRE( ” ” )
30   END
```

（メモリ番地の上限を＆HE000、文字領域の大きさを300バイトに設定した後、メモリの未使用領域並びに、未使用の文字領域の大きさを画面に表示します。）

EOF関数
（エンド オブ ファイル）

機能　ファイルが終了したかどうかを判定します。

書式　EOF（〈ファイル番号〉）

解説

〈ファイル番号〉で指定した入力ファイルが終了している場合は、−1を、そうでなければ0を与えます。〈ファイル番号〉はOPEN命令ですでに開かれていなければなりません。

サンプルプログラム

```
10   OPEN ”CAS : TEST” FOR OUTPUT AS #1
20   PRINT#1, 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 0
30   CLOSE
40   END
50   MAXFILES=2
60   OPEN ”CAS : TEST” FOR INPUT AS  #1
70   OPEN ”CRT : ” FOR OUTPUT AS #2
80   INPUT#1, A$
90   PRINT#2, A$
100  IF EOF (1)=−1 THEN PRINT EOF(1)
110  CLOSE : END
```

（行番号10～40：カセットレコーダに1～0までの数値データを記録します。行番号50～110：カセットレコーダから行番号40までのプログラムで記録したデータを読み込み、変数A$に代入し、画面に表示します。ファイルが終了すると、−1を表示します。）

ERL関数
(エラー ライン)

機能　**エラーの発生した行番号を与えます。**

書式　ERL

解説

エラーが発生したときの行番号を与えます。コマンド待ちの状態でエラーが発生したときは、65535を与えます。エラー回復処理ルーチンで、どの行でエラーが発生したかを調べるのに便利です。

注)IF文でERL関数の値を調べるとき、行番号は等号の右に書いてください。左に書くと、RENUMコマンドを実行したとき行番号が修正されません。

サンプルプログラム

```
10  ON ERROR GOTO 60
20  A=25
30  B$="YOKO"
40  PLINT A, B$
50  END
60  IF ERR=2 THEN PRINT ERL
70  RESUME 50
```

(エラー(この場合、Syntax error)が発生した行を画面に表示します。)

ERR関数
(エラー)

機能　**発生したエラーのエラーコードを与えます。**

書式　ERR

解説

エラーが発生したときのエラーコードを与えます。エラー回復処理ルーチンで、どのようなエラーが発生したかを調べるのに便利です。

エラーコードとその意味は、「エラーメッセージ一覧表」を参照してください。

CSRLIN関数
（カーソル ライン）

機能　画面上のカーソルの垂直位置を与えます。

書式　CSRLIN

解説

現在のカーソルの垂直位置を0～22（KEY OFFで0～23）の行単位で与えます。引数はありません。

サンプルプログラム

```
10  CLS
20  LOCATE  1, 7
30  PRINT CSRLIN
40  END
```

（カーソルの垂直位置（この場合は7）を画面に表示します。）

LPOS関数
（エル ポジション）

機能　プリンタヘッドの現在位置を与えます。

書式　LPOS（<引数>）

解説

プリンタバッファにおけるプリンタヘッドの現在の水平方向の位置を与えます。
<引数>は意味を持ちませんが、1文字記入が必要です。

サンプルプログラム

```
10  LPRINT "ATOSYUAZMUAKI";
20  PRINT "げんざいのヘッドのいちは"; LPOS
    (0)
30  END
```

（行番号10の文字列をプリンタに印字した後、プリンタヘッドの位置を画面に表示します。）

PAD関数
（パッド）

機能　タッチパッド、ライトペンなどの状態を返します。

書式　PAD（<引数>）

解説

指定した<引数>の値により、右表のような、タッチパッドの状態を返します。

PAD（0）またはPAD（4）が評価されたときにはじめて、そのタッチパッドの座標値が有効になりますので、2個のタッチパッドを同時に接続して使用した場合、ジョイスティック1に接続したタッチパッドの座標値に悪影響を及ぼすことがあります。

そのため、2つのタッチパッドの値を連続して読むときには2つのPAD関数の間に何行かのプログラムを入れるか、待ちループを入れます。

なお引数が7を越える場合は以下のような入力装置の状態を返します。

PAD（8）～PAD（11）：ライトペン

PAD（12）～PAD（15）：ポート1のマウスまたはトラックボール

PAD（16）～PAD（19）：ポート2のマウスまたはトラックボール

ライトペンの場合

引数の値	返される値の内容	返される値
8	ライトペンのデータが 有効 無効	 －1 0
9	ライトペンのX座標	座標値
10	ライトペンのY座標	座標値
11	ライトペンのスイッチが押されている 押されていない	－1 0

タッチパッドの場合

引数の値	タッチパッドの接続されたジョイスティックポート	返される値の内容	返される値
0	1	タッチパットに触れている 触れていない	－1 0
1	1	タッチパッドのX座標	座標値
2	1	タッチパッドのY座標	座標値
3	1	タッチパッドのスイッチが押されている 押されていない	－1 0
4	2	タッチパッドに触れている 触れていない	－1 0
5	2	タッチパッドのX座標	座標値
6	2	タッチパッドのY座標	座標値
7	2	タッチパッドのスイッチが押されている 押されていない	－1 0

マウス及びトラックボールの場合

引数の値	マウス・トラックボールの接続されたジョイスティックポート	返される値の内容	返される値
12	1	マウス・トラックボールに触れている 触れていない	−1 0
13	1	マウス・トラックボールのX座標	座標値
14	1	マウス・トラックボールのY座標	座標値
15	1	無意味	0

引数の値	マウス・トラックボールの接続されたジョイスティックポート	返される値の内容	返される値
16	2	マウス・トラックボールに触れている 触れていない	−1 0
17	2	マウス・トラックボールのX座標	座標値
18	2	マウス・トラックボールのY座標	座標値
19	2	無意味	0

PDL関数
(パドル)

機能　パドルの値を返します。

書式　PDL (<引数>)

解説

<引数>は、1から12の範囲で、 パドルの接続されたポートを指定します。<引数>が奇数のときは、ジョイスティック1に接続されたパドル、偶数のときは、ジョイスティック2に接続されたパドルの値を返します。

PLAY関数
(プレイ)

機能　**指定したPSGチャンネルがPLAY命令を実行中であるか否かを数値で返します。**

書式　**PLAY (〈チャンネル番号〉)**

文例　**X＝PLAY(3):PRINT X**

解説

〈チャンネル番号〉で指定したチャンネルが、PLAY命令を実行中である場合には、－1、そうでない場合には、0を返します。
〈チャンネル番号〉は、0～3の範囲で指定し、0が指定されると、全てのチャンネルについてチェックし、どれか1つでも実行中の場合には、－1を返します。

注) PLAY命令のすぐ後に、PLAY関数を入れると、現在の状態にかかわらず、－1を返すことがあります。

サンプルプログラム

```
10  PLAY "CDEFGAB"
20  X=PLAY (0)
30  PRINT X
40  END
```

(行番号10で音を発生させ、PLAY命令を実行中であることを画面に表示します。)

POINT関数
(ポイント)

機能　**グラフィック座標の点のカラーコードを調べます。**

書式　**POINT (〈水平位置〉,〈垂直位置〉)**

解説

画面上の指定した位置のカラーコードを0～15で与えます。

サンプルプログラム

```
10  SCREEN 2
20  PSET (200, 90), 1
30  PSET (100, 10), 8
40  A=POINT (200, 90)
50  B=POINT (100, 10)
60  SCREEN 1
70  PRINT A, B
80  END
```

(グラフィック画面に行番号20, 30で打たれた点の色を、テキスト画面にカラーコードで表示します。)

POS関数
（ポジション）

機能　**画面上のカーソルの水平位置の値を与えます。**

書式　POS（〈引数〉）

解説

〈引数〉を「0」にすると、画面上でのカーソルの水平位置を与えます。POS(0)が持つ値の範囲は、0～28です。(SCREEN 0を指定すると、0～38)〈引数〉は意味を持ちませんが、1文字記入が必要です。

サンプルプログラム

```
10  CLS
20  LOCATE 20, 20
30  PRINT POS(0)
40  END
```

（カーソルを20，20の位置に移動させた後、カーソルの水平位置を画面に表示します。）

STICK関数
（スティック）

機能　**指定したカーソルキーまたは、ジョイスティックの方向を0～8の数値で返します。**

書式　STICK（〈ジョイスティック番号〉）

解説

〈ジョイスティック番号〉は0から2の数値で、使用するジョイスティック、またはカーソルコントロールキーを指定します。

ジョイスティック番号	使用するジョイスティック
0	カーソルコントロールキー(をジョイスティックとして使用する場合)
1	ジョイスティック1
2	ジョイスティック2

ジョイスティックまたは、カーソルコントロールキーの方向により、次の値を返します。

サンプルプログラム

```
10  PRINT "カーソルコントロールキーをおして
    ください"
20  X=STICK(0)
30  PRINT X
40  GOTO 20
50  END
```

（押されたカーソルコントロールキーの方向を画面に、数値で表示します。）

1
8　2
7　0　3
6　4
5

STRIG関数
(エス トリガ)

機能　ジョイスティックのトリガボタンの状態を返します。

書式　STRIG (〈ボタン番号〉)

解説

〈ボタン番号〉で指定したジョイスティックのトリガボタンの状態を返します。
トリガボタンが押されると、－1を返します。
それ以外では0を返します。
〈ジョイスティックの番号〉は次のように指定します。

ジョイスティック番号	使用するジョイスティック
0	トリガボタンの代わりにキーボードのスペースキーを使う
1	ポート1につながっているジョイスティックのトリガボタン1
2	ポート2につながっているジョイスティックのトリガボタン1
3	ポート1につながっているジョイスティックのトリガボタン2
4	ポート2につながっているジョイスティックのトリガボタン2

ただし、ジョイスティックによってはトリガボタン2がないものもあります。
なお、トリガボタンが押されたときに、割込みがかかるように設定するON STRIG GOSUB命令もありますので、組合わせて使うと便利です。

サンプルプログラム

```
10  PRINT "スペースバーをおしてください"
20  X=STRIG(0)
30  PRINT X
40  GOTO 20
50  END
```

(スペースバーを押すと－1、それ以外のときは0を画面に表示します。)

特殊変数

TIME
(タイム)

➡ 基礎編119ページ

機能　内蔵クロックのセットおよびクロックの 1/60 秒ごとのカウント数を与えます。

書式　① TIME
② TIME ＝<式>

解説

①TIME

内蔵クロックのカウント数を与えます。値は、1/60秒単位です。ただし割り込みを用いる処理（一連の割り込み関係の命令や、カセットへの入出力）によって、多少の時間のずれを生じます。

②TIME＝<式>

内蔵クロックを<式>の値にセットしなおします。

<式>の値は0～65535まで使用できます。

サンプルプログラム

```
10  TIME=0 : PRINT TIME
20  FOR N=1 TO 100
30  NEXT N
40  PRINT TIME
50  GOTO 20
60  END
```

(経過した時間を 0 から画面に表示します。)

SPRITE$
（スプライト ドル）

➡ 基礎編103ページ

機能 スプライトのパターンを作ります。

書式 SPRITE$（〈パターン番号〉）＝〈文字式〉

解説

(〈文字式〉)で表わされるパターンを〈パターン番号〉で指定したスプライトに定義します。)

〈パターン番号〉は、SCREEN命令で指定したスプライトサイズが0、1の場合 0から255まで、スプライトサイズが2、3の場合は0から63まで指定できます。

〈文字式〉は、スプライト上のドット(点)を表わす2進数をキャラクターコードとする文字列を指定します。 下図のような図形をパターン番号1のスプライトとする場合

2進	16進
&B00000000	&H00
&B00000000	&H00
&B00111100	&H3C
&B01111110	&H7E
&B11011011	&HDB
&B11111111	&HFF
&B11111111	&HFF
&B01011010	&H5A

SPRITE$(1)＝CHR$(&H00)＋CHR$(&H00)＋CHR$(&H3C)＋CHR$(&H7E)＋CHR$(&HDB)＋CHR$(&HFF)＋CHR$(&HFF)＋CHR$(&H5A)

と指定します。

8×8スプライトでは8文字ぶん、16×16スプライトでは32文字ぶんの文字列を指定しなければなりませんが、それに満たない部分は0で満たされます。

16×16スプライトでは、下図のような順で文字列を指定します。ドット指定は、8ドット×8ドットの場合と同じです。

① ② ⋮ ⑧	⑰ ⑱ ⋮ ㉔
⑨ ⑩ ⋮ ⑯	㉕ ⋮ ㉜

VDP
（ブイディーピー）

機能 VDP（ビデオ信号IC)のレジスタの値を設定します。

書式 VDP（〈引数〉）

文例 VDP（1）＝&HE 1

(VDPのレジスタ1に”&HE1”というデータを書き込みます。)

解説

VDPはパラメーター−1〜−9，0〜24および33〜48に対応するレジスタにデータを書き込みまたは読み出します。

VDP、V−9938の規格および特殊変数VDP、BASEの働きを正確に理解してからお使いください。

BASE
(ベース)

機能　VDPレジスタの各テーブルの先頭番地を設定します。

書式　BASE(〈引数〉)

解説

〈引数〉で指定される VDP レジスタ内の各テーブルの先頭番地を設定します。

0から19までの〈引数〉と各テーブルの対応はつぎのとおりです。

引数	テーブル
0	テキストモードのパターン名称テーブルの先頭番地
1	意味を持ちません
2	テキストモードのパターンジェネレーターテーブルの先頭番地
3	意味をもちません
4	意味をもちません
5	テキストモードのパターン名称テーブルの先頭番地
6	テキストモードのカラーテーブルの先頭番地
7	テキストモードのパターンジェネレーターテーブルの先頭番地
8	テキストモードのスプライト属性テーブルの先頭番地
9	テキストモードのスプライトパターンテーブルの先頭番地
10	ハイリゾリューションモードのパターン名称テーブルの先頭番地
11	ハイリゾリューションモードのカラーテーブルの先頭番地
12	ハイリゾリューションモードのパターンジェネレーターテーブルの先頭番地
13	ハイリゾリューションモードのスプライト属性テーブルの先頭番地
14	ハイリゾリューションモードのスプライトパターンテーブルの先頭番地
15	マルチカラーモードのパターン名称テーブルの先頭番地
16	意味をもちません
17	マルチカラーモードのパターンジェネレーターテーブルの先頭番地
18	マルチカラーモードのスプライト属性テーブルの先頭番地
19	マルチカラーモードのスプライトパターンテーブルの先頭番地

20以上の引数については、INT〔〈引数〉／5〕が画面モードを、(〈引数〉MOD 5)がテーブルの種類を表します。テーブルの種類は右図のとおりです。

〈引数〉MOD 5	テーブルの種類
0	パターン名称テーブル
1	カラーテーブル
2	パターンジェネレーターテーブル
3	スプライト属性テーブル
4	スプライトパターンテ[illegible]ル

従って、〈引数〉は0～44までが指定可能です。

画面モードが5から8の場合には、先頭番地にアクティブページの開始アドレスを加えた値がVRAMの絶対番地になります。

画面のプリンタ出力命令
タブレットからの入力命令

H3専用拡張BASIC命令

CALL HCOPY/CHCOPY
(コール ハード コピー)

機能 スクリーンモードが0，1，3以外に設定されているときの画面を白黒（15階調）でプリンタに出力します。
注）カラーで出力することはできません。

書式 CALL　HCOPYまたは_HCOPY　……モノクロプリンタの場合
CALL　CHCOPYまたは_CHCOPY　……カラープリンタの場合

解説

H3専用拡張BASIC命令です。
モノクロのプリンタを使用する場合はCALL HCOPY、カラープリンタを使用する場合はCALL CHCOPYを使います。いずれの場合もカラーで出力することはできません。
_はCALLの省略形です。［SHIFT］キーを押しながら［ー・ん］キーを押してください。
カラーパレットを使用している場合には、パレット番号をカラーコードとみなして、階調をつけます。
従って、プリンタで印刷されたものと画面の状態とは印象の異なる場合があります。
プリンタが正常に作動しないと（プリントダウン）約10秒後にエラー（Device I/O error）となります。
スクリーンモード3で作成した絵の画面をプリンタに出力することはできません。
プリンタ出力を途中でやめたいときには、［CTRL］+［STOP］を押してください。
なお、カラープリンタのときリボンは黒リボンであることを確認してご使用ください。

CALL SCOPY/CSCOPY/CDCOPY (コール セレクト コピー)

機能 スクリーンモードが0，1，3以外に設定されているときの画面の中で指定された色の部分だけをプリンタに白黒（階調なし）で出力します。
注）カラーで出力することはできません。

書式 CALL　SCOPYまたは_SCOPY（カラーコード，カラーコード，……）

CALL　CSCOPYまたは_CSCOPY（カラーコード，カラーコード，……）

CALL　CDCOPYまたは_CDCOPY（カラーコード，カラーコード，……）

文例 CALL　SCOPY（1，2，3，5，6，7，8，9，10，11，12，13，14）

CALL　CSCOPY（1，2，3，5，6，7，8，9，10，11，12，13，14）

CALL　CDCOPY（1，2，3，5，6，7，8，9，10，11，12，13，14）

(画面の色の部分のうち、カラーコード4（暗い青）とカラーコード15（白）以外の部分をプリンタに白黒で出力します。)

解説

H3専用拡張BASIC命令です。

モノクロのプリンタを使用する場合はCALL SCOPY、カラープリンタを使用する場合はCALL CSCOPY、CDCOPYを使います。いずれの場合もカラーで出力することはできません。

_はCALLの省略形です。[SHIFT]キーを押しながら[ー・ん]キーを押してください。

カラーコードは出力したい色を、「,」で区切って0以外の全てのコードを書いておかなければなりません。

カラーパレットを使用している場合には、パレット番号をカラーコードとみなします。

この場合、カラーコード0は省略できません。

CSCOPYとCDCOPYの違いはCSCOPYは黒リボン、CDCOPYはカラーリボンのためのものです。これが異なりますとうまくコピーがとれませんのでご注意ください。

HCOPYと異なり、画面の色による階調はつきません。

プリンタが正常に動作しないと（プリントダウン）約10秒後にエラー（Device I/O error）となります。

スクリーンモード3で作成した絵の画面をプリンタに出力することはできません。

プリンタ出力を途中でやめたいときには、[CTRL]+[STOP]を押してください。

CALL CCOPY
(コール カラー コピー)

機能 スクリーンモードが0，1，3，以外に設定されているときの画面をカラーでカラープリンタに出力します。
カラープリンタでのみ有効です。
注）モノクロプリンタでの動作は保証できません。

書式 CALL　CCOPYまたは＿CCOPY

解説

H 3 専用拡張BASIC命令です。
＿はCALLの省略形です。[SHIFT]キーを押しながら[ー・ん]キーを押してください。
カラープリンタで出力可能な色数に制限があるため、画面の色がそのまま印刷されるとはかぎりません。
場合によっては、画面とは異なる印象を与えることもありますのでご注意ください。
プリンタが正常に動作しないと（プリントダウン）約10秒後にエラー（Device I/O error）となります。
スクリーンモード 3 で作成した絵の画面をプリンタに出力することはできません。
プリンタ出力を途中でやめたいときには、[CTRL]＋[STOP]を押してください。

CALL TABON/TABOFF
(コール タブオン/ダブオフ)

機能 タブレットからの手書き文字入力を可能なモードにする、またはそのモードをやめます。

書式 CALL　TABONまたは_TABON

CALL　TABOFFまたは_TABOFF

解説

H3専用拡張BASIC命令です。

_はCALLの省略形です。 SHIFT キーを押しながら ー・ん キーを押してください。

スクリーンモード0と1でCALL　TABONを実行するとタブレットからの手書き文字入力が可能になります。

また手書き文字入力の機能を活かすためにSCREEN 1でご使用ください。

手書き文字入力では、キーボードから入力できるキャラクタは全て入力可能です。

キーボードから同時に入力することも可能です。

TABONの状態では、画面の下から4行目までを候補文字ならびに手書き文字トレースの表示領域として使用するため、縦18行(KEY　OFFで19行)がベーシックで使用可能な領域となります。

WIDTH命令で、1行当りの表示文字を29文字以外にした場合には、手書き文字候補表示領域が崩れてしまうことがあります。

テキストモード1の画面では、手書き文字トレースの表示はしません。

CALL　TABOFFを実行すると手書き文字入力はできなくなります。

注) CLEAR文などで、BASICのエリアを&HC500以下にはしないでください。

付録（ふろく）

サンプルプログラム

1　カラー調整

```
10 ON STOP GOSUB 260
20 COLOR 15,1
30 SCREEN 2
40 STOP ON
50 OPEN "GRP:"AS #1
60 FOR Y=1 TO 3
70 FOR X=1 TO 5
80 C=(Y-1)*5+X
90 CIRCLE(40*X,48*Y+Z),15,C
100 IF X=1 AND Y=1 THEN CIRCLE(40,48),15
110 PAINT(40*X,48*Y+Z),C
120 NEXT X
130 Z=Z+10
140 NEXT Y
150 DRAW"BM38,20"
160 PRINT#1,"1␣␣␣␣2␣␣␣␣3␣␣␣␣4␣␣␣␣5"
170 DRAW"BM38,78"
180 PRINT#1,"6␣␣␣␣7␣␣␣␣8␣␣␣␣9␣␣␣␣10"
190 DRAW"BM36,135"
200 PRINT#1,"11␣␣␣12␣␣␣13␣␣␣14␣␣␣15"
210 DRAW"BM85,5"
220 PRINT #1,"カラー␣ちょうせい"
230 CLOSE
240 GOTO 240
250 END
260 COLOR 15,4,7
270 STOP OFF:RETURN 250
```

15色の円が描かれ、その上にカラーコードが表示されます。テレビの色調整つまみなどで、正しい色になるように画面の色を調節してください。
終わるときは、[CTRL] + [STOP] キーです。

2 カラーデモ1

```
10 SCREEN 2
20 FOR Y=0 TO 127 STEP 10
30 FOR X=0 TO 192 STEP 8
40 LINE(X,Y)-(255-X,191-Y),C,BF
50 C=C+1
60 IF C=15 THEN C=1
70 NEXT X,Y
80 GOTO 20
90 END
```

色紙を重ねるように、カラーコードの1から15までの色を次々に表示します。画面中央までくると今度は1枚1枚を貼がしていきます。重ねたり貼がしたりするたびに色紙は小さくなっていきます。

また、STOP キーを押すと変化が止まり、もう一度 STOP キーを押すとまた変化します。

終るときは CTRL + STOP キーです。

3 カラーデモ2

```
10 ON STOP GOSUB 220
20 COLOR 15,1
30 SCREEN 2
40 STOP ON
50 LINE(125,0)-(125,191)
60 LINE(0,95)-(255,95)
70 FOR Y=0 TO 40 STEP .25
80 PX=255*RND(1):PY=191*RND(1)
90 C=15*RND(1)
100 PSET(PX,PY),C
110 NEXT Y
120 FOR Y=0 TO 58 STEP 2
130 R=50*SIN(2*Y*3.14/191)
140 C=15*RND(1)
150 CIRCLE(125,Y),R,C,,,.4
160 CIRCLE(125,191-Y),R,C,,,.4
170 CIRCLE(19+Y,95),R,C,,,.5
180 CIRCLE(231-Y,95),R,C,,,.5
190 NEXT Y
200 GOTO 200
210 END
220 COLOR 15,4,7
230 STOP OFF:RETURN 210
```

4つに区切られた画面に、星のような点があらわれ、その後に4つのだ円が色を変えながら描かれます。

終わるときは CTRL + STOP キーです。

4 カラーデモ3

```
10 ON STOP GOSUB 390
20 COLOR 15,1:T=0
30 SCREEN 2
40 STOP ON
50 C1=15*RND(SIN(T))
60 IF C1=1 OR C1=0 THEN GOTO 50
70 PX=120*RND(1):PY=160*RND(1):R=1
80 X=PX+80:Y=PY
90 C=C1:GOSUB 340
100 C=C1:GOSUB 360
110 PX=PX+8
120 R=R+1
130 IF R<=10 GOTO 90
140 PX=PX-80:R=1
150 C=1:GOSUB 340
160 C=C1:GOSUB 360
170 PX=PX+8
180 R=R+1
190 IF R<=10 GOTO 150
200 PX=PX+8
210 C=1:GOSUB 360
220 C=C1:GOSUB 340
230 PX=PX+8
240 R=R-1
250 IF R>=1 THEN GOTO 210
260 PX=PX-88:R=10
270 C=1:GOSUB 340
280 C=1:GOSUB 360
290 PX=PX+8
300 R=R-1
310 IF R>=0 GOTO 270
320 T=T+1:IF T>1000 THEN T=0
330 GOTO 50
340 CIRCLE(PX,PY),5,C,,,.7
350 RETURN
360 CIRCLE(X,Y),4*R,C,,,2
370 RETURN
380 END
390 COLOR 15,4,7
400 STOP OFF:RETURN 380
```

大きくなったり小さくなったりする円の中を、くさりが通り抜けてゆきます。CTRL + STOP キーが押されるまでどんどん描かれます。

5 UFO(ユーフォー)の着陸(ちゃくりく)

```
10 COLOR 15,4,7
20 ON STOP GOSUB 720
30 SCREEN 2,2
40 STOP ON
50 LINE(0,120)-(255,191),2,BF
60 CIRCLE(30,150),30,15,,,.5
70 LINE(60,150)-(0,150)
80 X=X+.5
90 IF X>=200 THEN X=0
100 LINE(45,150-7.5*SQR(3))-(15,150+7.5*SQR(3))
110 COLOR 6
120 DRAW"BM60,160R30M70,180R40M130,160R200"
130 DRAW"BM60,160M80,140R30M120,130R40M150,140R200"
140 PAINT(120,150),6
150 LINE(75,145)-(255,145),15
160 LINE(70,155)-(255,155),15
170 LINE(130,130)-(80,180),15
180 LINE(150,130)-(100,180),15
190 CIRCLE(172,180),6,6,,,.5
200 PAINT(172,180),6
210 LINE(172,130)-(172,180),15
220 COLOR 5
230 DRAW"BM140,125M+20,+2R7E3M+10,-4M+7,+12M+20,-5R50"
240 DRAW"BM140,125M+12,-20E30R10F10R5M+10,-30E10R5F15E40"
250 PAINT(180,100)
260 COLOR 12
270 DRAW"BM100,120M-12,+8L10M-8,+2M-7,-2H7L5M-8,+5L2M-20,
    -5M-20,+8M-10,-3L20"
280 C=12
290 DRAW"BM100,120M-12,-6H25L10G4L5M-15,-20H10L5M-8,+5L30"
300 PAINT(60,100),C
310 FOR J=1 TO 11
320 B$=""
330 READ A$
340 FOR I=1 TO 32
350 B$=B$+CHR$(VAL("&H"+MID$(A$,2*I-1,2)))
360 NEXT I
370 SPRITE$(J)=B$
380 NEXT J
390 FOR K=0 TO 1
400 PUT SPRITE 1+K*10,(80+80*K,50-30*K),15,1
410 PUT SPRITE 2+K*10,(80+80*K,66-30*K),15,2
420 PUT SPRITE 3+K*10,(96+80*K,50-30*K),15,3
430 PUT SPRITE 4+K*10,(96+80*K,66-30*K),15,4
440 NEXT K
450 PUT SPRITE 5,(170,118),1,11
```

```
460 PUT SPRITE 6,(170,118),11,10
470 X=230:Y=100
480 Y=140-X*X/1000
490 PUT SPRITE 7,(X,Y),C,5
500 PUT SPRITE 8,(X+16,Y),C,6
510 C=INT(RND(1)*15)
520 X=X-.5:IF X>=15 THEN GOTO 480
530 X=0:Y=0:K=0
540 PUT SPRITE 9,(27+X,130-Y),1,9
550 Y=Y+3*RND(1):IF Y<75 THEN GOTO 540
560 X=X+10:Y=90+20*SIN(.06*X+RND(1))
570 IF K=5 THEN PUT SPRITE 9,(27+X,130-Y),0,9:GOTO 470
580 IF X>255 THEN X=0:K=K+1:GOTO 540
590 GOTO 540
600 DATA 00000000000000000000010307777FFF00000000000000000107CF
    EFFFFFFFFF
610 DATA FF7FFFFF793000000000000000000000FFFFFFFFFFBF1F0F060000
    0000000000
620 DATA 000000000000000080ECFEFFFFFFFFFFF000000000000000183C7E
    7FFEFCFFFF
630 DATA FFFFFFFFFFFBF1C00000000000000000FFFEFEFFFEFCFEFCF87000
    0000000000
640 DATA 000000000102040C1F3F03030100000001070FFFFF929292FFFFF0
    F0E0C00000
650 DATA 80E0F0FFFF494949FFFF0F0F070300000000000080402030F8FCC0
    C080000000
660 DATA 000000000102040C1F3F00000000000001070FFFFF929292FFFF07
    0703010000
670 DATA 80E0F0FFFF494949FFFFE0E0C08000000000000080402030F8FC00
    0000000000
680 DATA 247E9999997E24C300000000000000000000000000000000000000
    0000000000
690 DATA A3C3FFFFFFFFFFFFFFFFFFFFFF7F3800C0F8F7FFFFFFFFFFFFFFFF
    FFFF1F0700
700 DATA 000007722222223E222222227700000000000083808080808080808
    082C000000
710 END
720 COLOR 15,4,7
730 STOP OFF:RETURN 710
```

滑走路を通ってUFOが着陸します。停止すると中からインベーダーが飛び出してきて、空を跳びはねます。左の雲の後を通り、右の雲の前を通り過ぎるところに注目してください。このくり返しを終了させたいときは、[CTRL] ＋ [STOP] キーを押します。

6 スプライトパターン作成プログラム

```
10 ON STOP GOSUB 500
20 STOP ON
30 CLS
40 PRINT"␣␣␣␣␣␣〈ﾊﾟﾀｰﾝ さくせい〉"
50 PRINT
60 PRINT"␣␣␣␣0123456789ABCDEF"
70 PRINT
80 FOR I=0 TO 15
90 PRINT"␣␣";HEX$(I);"␣················"
100 NEXT I
110 LOCATE 20,6:PRINT"␣いどうは"
120 LOCATE 20,7:PRINT"␣ｶｰｿﾙｷｰで"
130 LOCATE 20,9:PRINT"␣ｾｯﾄは"
140 LOCATE 20,10:PRINT"␣Sｷｰで"
150 LOCATE 20,12:PRINT"␣ﾘｾｯﾄは"
160 LOCATE 20,13:PRINT"␣Rｷｰで"
170 LOCATE 20,15:PRINT"␣ｴﾝﾄﾞは"
180 LOCATE 20,16:PRINT"␣Eｷｰで"
190 PX=0:PY=0
200 A$=INKEY$
210 LOCATE PX+4,3:PRINT"◆"
220 LOCATE 3,PY+4:PRINT"◆"
230 IF A$=CHR$(28) AND PX<15 THEN GOSUB 510:PX=PX+1
240 IF A$=CHR$(29) AND PX>0  THEN GOSUB 510:PX=PX-1
250 IF A$=CHR$(31) AND PY<15 THEN GOSUB 510:PY=PY+1
260 IF A$=CHR$(30) AND PY>0  THEN GOSUB 510:PY=PY-1
270 IF A$="s" OR A$="S" THEN LOCATE PX+4,PY+4:PRINT"●"
280 IF A$="r" OR A$="R" THEN LOCATE PX+4,PY+4:PRINT"·"
290 IF A$="e" OR A$="E" THEN GOTO 320
300 GOTO 200
310 B=0
320 LOCATE PX+4,3:PRINT"␣":LOCATE 3,PY+4:PRINT"␣":FOR J=0 TO 15
330 A=0
340 FOR I=B TO B+7
350 A=A*2
360 P$=CHR$(VPEEK(6144+32*(J+4)+I+6))
370 IF P$="●" THEN A=A+1
380 NEXT I
390 Q$=RIGHT$("00"+HEX$(A),2)
400 K$=K$+Q$
410 NEXT J
420 B=I:IF B=8 THEN GOTO 320 ELSE 430
430 CLS
440 FOR I=0 TO 15
450 X$(1)=MID$(K$,2*I+1,2)
460 X$(2)=MID$(K$,33+2*I,2)
470 PRINT X$(1),X$(2)
480 NEXT I
```

```
490 END
500 CLS:STOP OFF:RETURN 490
510 LOCATE PX+4,3:PRINT"␣"
520 LOCATE 3,PY+4:PRINT"␣"
530 RETURN
```

たて16個、よこ16個の「・」を表示します。カーソルコントロールキー(◁▷▽△)を使ってたてとよこの位置を決めて、Sキーを押します。「・」は白丸になりますから、これをくり返して作りたいスプライトパターンを描いていきます。Sキーを押し間違えたり、訂正したいときには、Rキーを押します。スプライトパターンを作り終ったら、Eキーを押します。しばらくすると画面には、16×16ドットのスプライトパターンを16進数に変換した値で表示します。106ページのように、左上の数字から文字列に変換していけばよいのです。左上の1/4の8×8個でパターンを作成すれば、8×8ドットのスプライトも作れます。

7 おいかけっこ

```
10 KEY OFF
20 ON STOP GOSUB 500
30 STOP ON
40 DEFINT A-Z
50 MX=28:MY=23:DX=1:DY=1:NN=2:DL=500
60 XX=INT(RND(1)*MX):YY=INT(RND(1)*MY)
70 X=INT(RND(1)*MX):Y=INT(RND(1)*MY)
80 CLS:LOCATE 0,23:PRINT"SCORE";SC;
90 TIME=0
100 DR=INT(RND(1)*8):LOCATE XX,YY:PRINT"␣";
110 IF DR=0 THEN DX=-DX:GOTO 140
120 IF DR=1 THEN DY=-DY:GOTO 140
130 IF DR=2 THEN DX=-DX:DY=-DY
140 XX=XX+DX:YY=YY+DY
150 IF XX>=MX THEN XX=MX-1
160 IF YY<0 THEN YY=0
170 IF YY>=MY THEN YY=MY-1
180 IF XX<0 THEN XX=0
190 LOCATE XX,YY:PRINT"@";
200 FOR II=1 TO DL:NEXT II
210 FOR JJ=1 TO NN:LOCATE X,Y:PRINT"␣";
220 N=STICK(1)
230 IF 1=N THEN Y=Y-ABS(DY):GOTO 360
240 IF 7=N THEN X=X-ABS(DX):GOTO 360
250 IF 5=N THEN Y=Y+ABS(DY):GOTO 360
260 IF 3=N THEN X=X+ABS(DX):GOTO 360
270 IF 2=N THEN X=X+ABS(DX):Y=Y-ABS(DY):GOTO 360
280 IF 4=N THEN X=X+ABS(DX):Y=Y+ABS(DY):GOTO 360
290 IF 6=N THEN X=X-ABS(DX):Y=Y+ABS(DY):GOTO 360
300 IF 8=N THEN X=X-ABS(DX):Y=Y-ABS(DY):GOTO 360
310 N$=INKEY$
320 IF N$=CHR$(&H1E) THEN Y=Y-ABS(DY):GOTO 360
330 IF N$=CHR$(&H1D) THEN X=X-ABS(DX):GOTO 360
340 IF N$=CHR$(&H1F) THEN Y=Y+ABS(DY):GOTO 360
350 IF N$=CHR$(&H1C) THEN X=X+ABS(DX)
360 IF X>=MX THEN X=MX-1:GOTO 400
370 IF X<0 THEN X=0:GOTO 400
380 IF Y>=MY THEN Y=MY-1:GOTO 400
390 IF Y<0 THEN Y=0
400 LOCATE X,Y:PRINT"♥"
410 IF (X=XX) AND (Y=YY) THEN GOTO 440
420 NEXT JJ
430 GOTO 100
440 LOCATE 15,10:PRINT"CATCH"
450 S=TIME
460 SC=10000/S+SC:DL=DL-20
470 FOR I=1 TO 500:NEXT I
480 GOTO 60
```

```
490 END
500 CLS:KEY ON
510 STOP OFF:RETURN 490
```

画面には、「@」(アットマーク)と「♥」(ハートマーク) が出ます。あなたは、♥マークです。カーソルコントロールキー (◁▷▽△) を使って、@マークを捕まえてください。(2つのキーを同時に押すと斜め方向に動きます。)うまくつかまえると、「CATCH」と表示します。スコアは時間制ですから、早くつかまえればつかまえるほどよい得点になります。
ジョイスティック(別売)をお持ちの人は、ジョイスティック接続コネクタ1に接続すれば使えます(斜め方向も使えます)。

終了するときは、 CTRL + STOP キーを押します。

8 インベーダーダンス

```
10 COLOR 15,4,7
20 SCREEN2,0
30 I=1
40 SPRITE$(1)=CHR$(&H24)+CHR$(&H7E)+CHR$(&H99)+CHR$(&H99)+
   CHR$(&H99)+CHR$(&H7E)+ CHR$(&H24)+CHR$(&HC3)
50 LINE(0,96+I)-(255,96+I)
60 LINE(0,96-I)-(255,96-I)
70 LINE(127,95)-(255-I,191)
80 LINE(127,95)-(I,0)
90 LINE(127,95)-(I,191)
100 LINE(127,95)-(255-I,0)
110 LINE(127,95)-(255,95+I)
120 LINE(127,95)-(255,95-I)
130 LINE(127,95)-(0,95+I)
140 LINE(127,95)-(0,95-I)
150 I=I*1.4
160 IF I<115 THEN GOTO 50
170 R=R+1
180 CIRCLE(127,95),R,1,,,.2
190 IF R<80 GOTO 170
200 LINE(0,76)-(255,115),1,BF
210 X=X+6+9*RND(1):Y=10*SIN(.5*X)
220 PUT SPRITE 3,(X,Y+95),3,1
230 IF X>255 THEN X=0
240 GOTO 210
250 END
```

奥行きを感じさせる画面の中央を、インベーダーが跳びはねます。終了させたいときは、CTRL + STOP キーを押します。

9 年賀状のプリント

```
1 CLEAR 200,&HF37F
100 CLS
110 COLOR 1,15,15:SCREEN 2
120 LINE(71,20)-(176,180),1,B
130 OPEN "GRP:" AS #1
140 X= 90:Y=50:DRAW"BM=X;,=Y;"
150 PRINT #1,"あけましで"
160 X=100:Y=65:DRAW"BM=X;,=Y;"
170 PRINT #1,"おめて゛とう"
180 X=110:Y=80:DRAW"BM=X;,=Y;"
190 PRINT #1,"こ゛さ゛います"
200 COLOR 4
210 COLOR 8
220 X=115:Y=105:DRAW"BM=X;,=Y;"
230 PRINT #1,"1月 1日"
240 LINE(72,157)-(175,179),9,BF
250 LINE( 72,157)-(175,157),6
260 LINE( 72,170)-( 95,170),6
270 LINE(100,179)-(140,130),3
280 LINE-(150,130),3
290 LINE-(175,170),3
300 LINE-(175,179),3
310 LINE-(100,179),3
320 PAINT(150,150),3
330 CALL HCOPY
340 COLOR 15,4,7
400 END
```

年賀状を画画に描いた後、プリンタに画面のハードコピーを出力します。絵や、あいさつを変えて“暑中見舞い”“招待状”など、くふうしだいでいろんなものをプリントできます。

コントロール コード一覧表

CTRL キーを押しながら以下のキーを押します（DELは除く）。

キー	コード		同じ機能のキー	機　能
	10進	16進		
B *	2	0 2		カーソルを1つ前の語の先頭に移動 (語の途中で押した場合はその語の先頭に移動)
C *	3	0 3		入力待ちの状態を中断する
E *	5	0 5		その行のカーソル以後を消去
F *	6	0 6		カーソルを次の語の先頭に移動
G *	7	0 7		ブザーをならす
H	8	0 8	BS	カーソルを1文字戻し、そこにあった文字を消去
I	9	0 9	TAB	次のタブストップまでカーソルを移動
J *	10	0 A		ライン・フィード(次の行にカーソルを移動)
K *	11	0 B	HOME	カーソルを画面左上に移動
L *	12	0 C	CLS	全画面を消去してカーソルを左上すみに移動
M *	13	0 D	RETURN	キャリッジ・リターン(1行の終了)
R *	18	12	INS	インサート・モードの切り換え
U *	21	15		その1行を消去
X *	24	18	SELECT	
¥ *	28	1 C	>	カーソルを右に移動
] *	29	1 D	<	カーソルを左に移動
^ *	30	1 E	∧	カーソルを上に移動
_ *	31	1 F	∨	カーソルを下に移動
DEL	127	7 F	DEL	カーソル位置の文字を消去します

注）「*」マークのついたコントロール・コードを入力すると、インサート・モードは通常モードに復帰します。

注）CLSキーは SHIFT キーを押しながら入力します。

キャラクターコード一覧表／カラーコード一覧表

●キャラクターコード一覧表

2バイトコード
（上位1バイト01H）

下位4ビット＼上位4ビット	4	5
0		π
1	月	┴
2	火	┬
3	水	┤
4	木	├
5	金	┼
6	土	│
7	日	─
8	年	┌
9	円	┐
10	時	└
11	分	┘
12	秒	╳
13	百	大
14	千	中
15	万	小

1バイトコード

下位4ビット＼上位4ビット	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		0	@	P	'	p	♠			ー	タ	ミ	た	み
1	!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
2	"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
3	#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も
4	$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や
5	%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ
6	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
10	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
11	+	;	K	〔	k	¦	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
12	,	<	L	¥	l	¦	ゃ	し	ャ	シ	フ	ワ	ふ	わ
13	−	=	M	〕	m	¦	ゅ	す	ュ	ス	ヘ	ン	へ	ん
14	.	>	N	∧	n	~	ょ	せ	ョ	セ	ホ	゛	ほ	
15	/	?	O	_	o		っ	そ	ッ	ソ	マ	゜	ま	█

█ ←カーソル

●カラーコード一覧表

カラーコード	色	カラーコード	色
0	透明	8	赤
1	黒	9	明るい赤
2	緑	10	暗い黄
3	明るい緑	11	明るい黄
4	暗い青	12	暗い緑
5	明るい青	13	紫（マゼンタ）
6	暗い赤	14	灰
7	水色（シアン）	15	白

エラーメッセージ一覧表

エラーコード	エラーメッセージ	内　　容
1	NEXT without FOR	FOR～NEXTが正しく対応していない。(FOR文がないのにNEXT文に出会った。)
2	Syntax error	プログラムが文法にそっていない。(打ち間違いや書き方の間違いがある。)
3	RETURN without GOSUB	GOSUB～RETURNが正しく対応していない。(GOSUB文がないのにRETURNに出会った。)
4	Out of DATA	READすべきDATA文がない。(データを読みつくした後、さらにREAD文がある。)
5	Illegal function call	関数やステートメントの機能の呼び方が間違っている。
6	Overflow	計算結果や入力した数値がベーシックの数値形式の許容範囲をこえている。
7	Out of memory	メモリ容量が足りなくなった。(プログラムが長すぎる、ファイルが多すぎる、変数が多すぎるなど。)
8	Undefined line number	指定した行番号が存在しない。
9	Subscript out of range	配列の添字が規定の範囲内にない。(添字の数値が間違っている。)
10	Redimensioned array	同じ配列を二重に定義している。または初期値の10の配列が成立した後で、さらに配列を定義している。
11	Division by zero	0で割り算をしています。
12	I llegal direct	直接モードで使うことのできないコマンドを直接モードで実行しようとした。(DEFFN命令など)
13	Type mismatch	数値と文字列というように、変数または定数の型、関数の引数の型があわない。

エラーコード	エラーメッセージ	内容
14	Out of string space	文字変数用のメモリエリアがたりなくなった。
15	String too long	文字式の値の文字数が255を超えた。
16	String formula too complex	文字式が複雑すぎる。
17	Can't CONTINUE	Count 命令によるプログラムの続行ができない。(エラーにより実行が停止している、CTRL+STOPの後、プログラムの修正を行なった。または続行すべきプログラムがない。)
18	Undefined user function	DEF FNにより、まだ定義されていないユーザ関数が呼ばれた。
19	Device I /O error	カセットまたはプリンタ、ディスプレイに入出力エラーが発生した。ベーシックでは、このエラーから回復することはできません。
20	Verify error	カセットにセーブしたプログラムが現在のプログラムとは異なっている。
21	No RESUME	エラー処理ルーチンにRESUMEがない。
22	RESUME without error	エラーと RESUME 文が対応していない。(エラーが発生していないのにRESUMEを実行しようとした。)
23	Unprintable error	エラーメッセージの定義されていないエラーが発生した。
24	Missing operand	演算に必要な数値が指定されていない。(式の右辺がないなど)
25	Line buffer overflow	入力された行の文字数が多すぎる。
26 〜 49	Unprintable error	エラーの定義がされていませんが、ベーシック拡張の際使用されることが予約されています。

エラーコード	エラーメッセージ	内　　　　容
50	FIELD overflow	ランダムファイルの指定されたレコード長のバイト数を超えている、またはFIELDバッファがランダムファイルへのシーケンシャルI/O実行中が終わってしまった。
51	Internal error	ベーシック内部に異常が発生した。通常発生することはありませんが、万一生じた場合には、電源を一度切って入れなおしてください。
52	Bad file number	OPENされていないファイルのファイル番号を使用した。またはMAXFILE文により指定したファイル番号の範囲を超えたファイル番号のファイルを使用した。
53	File not found	指定されたファイルが見つからない。
54	File already open	すでにOPENされているファイルをさらにOPENしようとした。
55	Input past end	ファイルの全てのデータを読みつくした後に、Input命令を実行した。
56	Bad file name	ファイル名やデバイスディスクリプタのつけ方を間違えた。
57	Direct statement in file	ASCII 形式 ファイル中に、 直接モードの命令文がある。
58	Sequential I/O only	シーケンシャルファイルに対し、ランダムアクセス命令を実行した。
59	File not OPEN	OPENされていないファイルを使用しようとした。
60 ～ 255	Unprintable error	未定義のエラーコードです。ユーザが任意にエラーコードを定義して使用することができます。

（アルファベット順）

MSXベーシック索引

コマンド、ステートメント

（アルファベット順）

MSXベーシック索引

関数、特殊変数

（アルファベット順）

内蔵ソフト用コマンド索引

プリンタ出力命令・タブレットからの入力命令

MSXベーシック説明書

日立家電販賣株式會社
〒105 東京都港区西新橋2－15－12
電話(03)502-2111

株式會社 日立製作所
〒105 東京都港区西新橋2－15－12
電話(03)502-2111

MSX2(B)Ⓑ